Chapter 3
Super Learning

Eric C. Polley, Sherri Rose, Mark J. van der Laan

This is the first chapter in our text focused on estimation within the road map for targeted learning. Now that we've defined the research question, including our data, the model, and the target parameter, we are ready to begin. For the estimation of a target parameter of the probability distribution of the data, such as target parameters that can be interpreted as causal effects, we implement TMLE. The first step in this estimation procedure is an initial estimate of the data-generating distribution P_0, or the relevant part Q_0 of P_0 that is needed to evaluate the target parameter. This is the step presented in Chap. 3, and TMLE will be presented in Chaps. 4 and 5.

We introduce these concepts using our mortality study example from Chap. 2 examining the effect of LTPA. Our outcome Y is binary, indicating death within 5 years of baseline, and A is also binary, indicating whether the subject meets recommended levels of physical activity. The data structure in this example is $O = (W, A, Y) \sim P_0$. Our target parameter is $\Psi(P_0) = E_{W,0}[E_0(Y \mid A = 1, W) - E_0(Y \mid A = 0, W)]$, which represents the causal risk difference under causal assumptions. Since this target parameter only depends on P_0 through the conditional mean $\bar{Q}_0(A, W) = E_0(Y \mid A, W)$, and the marginal distribution $Q_{W,0}$ of W, we can also write $\Psi(Q_0)$, where $Q_0 = (\bar{Q}_0, Q_{W,0})$. We estimate the expectation over W with the empirical mean over W_i, $i = 1, \ldots, n$. With this target parameter, $\bar{Q}_0(A, W) = E_0(Y \mid A, W)$ is the only object we will still need to estimate. Therefore, the first step of the TMLE of the risk difference $\Psi(P_0)$ is to estimate this conditional mean function $\bar{Q}_0(A, W)$. Our substitution TMLE will be of the type

$$\psi_n = \Psi(Q_n) = \frac{1}{n} \sum_{i=1}^{n} \{\bar{Q}_n(1, W_i) - \bar{Q}_n(0, W_i)\},$$

where this estimate is obtained by plugging $Q_n = (\bar{Q}_n, Q_{W,n})$ into the parameter mapping Ψ.

We could estimate the entire conditional probability distribution of Y, instead of estimating the conditional mean of Y, but then (except when Y is binary) we are estimating portions of the density we do not need. Targeted estimation of only the

M.J. van der Laan and S. Rose, *Targeted Learning: Causal Inference for Observational and Experimental Data*, Springer Series in Statistics, DOI 10.1007/978-1-4419-9782-1_3,

relevant portion of the probability distribution of O in this first step of the TMLE procedure provides us with maximally efficient and unbiased estimators. This will be further discussed in Chaps. 4 and 5.

3.1 Background

Let's start our discussion with studies where Y is binary, such as in our mortality study example. When Y is binary, there is no difference between the conditional mean or conditional probability distribution, so this distinction plays no role. Now, what do we know about our probability distribution P_0 of O? We know that the data are n i.i.d. observations (realizations) on n i.i.d. copies $O_1, \ldots, O_n$ of $O \sim P_0$. These realizations are denoted $o_1, \ldots, o_n$. In our mortality study example, we have no knowledge about P_0. Thus we have a nonparametric statistical model for P_0. In scenarios where we have some knowledge about data generation, we can include this knowledge in a semiparametric statistical model. Our parameter of interest is this chapter is $\bar{Q}_0(A, W) = P_0(Y = 1 \mid A, W)$.

How are we to estimate $P_0(Y = 1 \mid A, W)$ if we assume only a nonparametric (or, in general, a large semiparametric) statistical model? We do not know anything about the shape of $\bar{Q}_0(A, W)$ as a function of exposure and covariates. Standard practice would assume a parametric statistical model, making assumptions we know are wrong, and proceeding to estimate $P_0(Y = 1 \mid A, W)$ under the assumptions of the parametric statistical model, thereby forcing the shape of this function of (A, W) to follow an incorrect user-supplied structure. Since the parametric statistical model is wrong, the estimate of $P_0(Y = 1 \mid A, W)$ will be biased, and increasing the sample size will not make it any better. What we want is an automated algorithm to nonparametrically (or semiparametrically) estimate $P_0(Y = 1 \mid A, W)$, i.e., we want an estimator that is able to learn from the data using the true knowledge represented by the actual statistical model for P_0.

In the computer science literature, this is called machine learning. In statistics, these methods are often referred to as nonparametric or semiparametric estimators, or data-adaptive estimators. We will use the terms *data-adaptive* and *machine learning* interchangeably in this text. The essential point is that there are nonparametric methods that also aim to "smooth" the data and estimate this regression function flexibly, adapting it to the data given a priori guidelines, without overfitting the data.

For example, one could use local averaging of the outcome Y within covariate "neighborhoods." Here, neighborhoods are bins for covariate observations that are close in value, where these bins are defined by partitioning the covariate space. The number of bins will determine the smoothness of our fitted regression function. Such a regression estimator is also called a histogram regression estimator. How do you choose the size of these neighborhoods or bins? This becomes a bias–variance trade-off question. If we have many small neighborhoods, the estimate will not be smooth and will have high variance since some neighborhoods will be empty or contain only a small number of observations. The result is a sample mean of the outcome

over the observations in the neighborhood that is imprecise. On the other hand, if we have very few large neighborhoods, the estimate is much smoother, but it will be biased since the neighborhoods fail to capture the complexity of the data. Suppose we choose the number of neighborhoods in a smart way. With n large enough, this will result in a good estimator in our nonparametric statistical model. Formally, we say such a histogram regression estimator is asymptotically consistent in the sense that it approximates the true regression function as sample size increases.

However, if the true data-generating distribution is very smooth, a logistic regression in a misspecified parametric statistical model might beat the nonparametric estimator. This is frustrating! We want to create a smart nonparametric estimator that is consistent, but in some cases it may "lose" to a misspecified parametric model because it is more variable. There are other ways of approaching the truth that will be smoother than local averaging. One method, locally weighted regression and scatterplot smoothing (loess), is a weighted polynomial regression method that fits the data locally, iteratively within neighborhoods. Spline functions, another method, are similar to polynomial functions, as splines are piecewise polynomial functions. Smoothing splines use penalties to adjust for a lack of smoothness, and regression splines use linear combinations of basis functions. There are many other potential algorithms we could implement to estimate $P_0(Y = 1 \mid A, W)$. However, how are we to know priori which one to use? We cannot bet on a logistic regression in a misspecified parametric statistical model, but we have the problem that one particular algorithm is going to do better than the other candidate estimators for the particular data-generating distribution P_0, and we do not know which one is the best.

To be very explicit, an algorithm is an estimator of $\bar{Q}_0$ that maps a data set of n observations (W_i, A_i, Y_i), $i = 1, \ldots, n$, into a prediction function that can be used to map input (A, 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 use *algorithm* in a general sense to mean any mapping from data into a predictor, so that the word *algorithm* is equivalent to the word *estimator*. As long as the algorithm takes the observed data and outputs a fitted prediction function, we consider it a prediction algorithm. For example, a collection of algorithms could include least squares regression estimators, algorithms indexed by set values of the fine-tuning parameters for a collection of values, algorithms using internal cross-validation to set fine-tuning parameters, algorithms coupled with screening procedures to reduce the dimension of the covariate vector, and so on.

Effect Estimation vs. Prediction

Both causal effect and prediction research questions are inherently *estimation* questions. In the first, we are interested in estimating the causal effect of A on Y adjusted for covariates W. For prediction, we are interested in generating a function to input the variables (A, W) and predict a value for Y. These are separate and distinct research questions. However, many (causal) effect esti-

mators, such as TMLE, involve prediction steps within the procedure. Thus, understanding prediction is a core concept even when one has an effect estimation research question. Effect parameters where no causal assumptions are made are often referred to as variable importance measures (VIMs).

3.2 Defining the Estimation Problem

Our data structure is $O = (W, A, Y) \sim P_0$, and we observe n i.i.d. observations on $O_1, \ldots, O_n$. An estimator maps these observations into a value for the parameter it targets. We can view estimators as mappings from the empirical distribution P_n of the data set, where P_n places probability $1/n$ on each observed O_i, $i = 1, \ldots, n$. In our mortality study example, we need an estimator of $\bar{Q}_0(A, W) = P_0(Y = 1 \mid A, W)$.

Before we can choose a "best" algorithm to estimate the function $\bar{Q}_0 : (A, W) \rightarrow \bar{Q}_0(A, W)$, we must have a way to define what "best" means. We do this in terms of a loss function, which assigns a measure of performance to a candidate function $\bar{Q}$ when applied to an observation O. That is, a loss function is a function L given by

$$L : (O, \bar{Q}) \rightarrow L(O, \bar{Q}) \in \mathbb{R}.$$

It is a function of the random variable O and parameter value $\bar{Q}$. Examples of loss functions include the L_1 absolute error loss function

$$L(O, \bar{Q}) = |Y - \bar{Q}(A, W)|,$$

the L_2 squared error (or quadratic) loss function

$$L(O, \bar{Q}) = (Y - \bar{Q}(A, W))^2,$$

and the negative log loss function for a binary Y

$$L(O, \bar{Q}) = -\log(\bar{Q}(A, W)^Y (1 - \bar{Q}(A, W))^{1-Y}).$$

A loss function defines a function $\bar{Q}_0$ that has the optimal expected performance with respect to that loss function among all candidate functions $\bar{Q}$. For example, the function $\bar{Q}_0$ that minimizes the expected absolute error, $\bar{Q} \rightarrow E_0|Y - \bar{Q}(A, W)|$, is the conditional median of Y, as a function of (A, W). On the other hand, the function that minimizes the expected squared error is the conditional mean of Y, while the function that minimizes the expected negative log loss function for a binary Y is the conditional probability distribution of Y, as a function of (A, W). For binary Y, both the L_2 loss and negative log loss target the same function $\bar{Q}_0(A, W) = P_0(Y = 1 \mid A, W)$.

We can now define our parameter of interest, $\bar{Q}_0(A, W) = E_0(Y \mid A, W)$, as the minimizer of the expected squared error loss:

$$\bar{Q}_0 = \arg\min_{\bar{Q}} E_0 L(O, \bar{Q}),$$

where $L(O, \bar{Q}) = (Y - \bar{Q}(A, W))^2$. $E_0 L(O, \bar{Q})$, which we want to be small, evaluates the candidate $\bar{Q}$, and it is minimized at the optimal choice of $\bar{Q}_0$. We refer to expected loss as the risk. Thus we have a way to define the "best" algorithm. We want the estimator of the regression function $\bar{Q}_0$ whose realized value minimizes the expectation of the squared error loss function. If we have two estimates $\bar{Q}_n^a$ and $\bar{Q}_n^b$, then we prefer the estimator for which $\sum_o P_0(O = o)L(o, \bar{Q}_n)$ is smallest.

This makes sense intuitively. We want an estimator that is close to the true $\bar{Q}_0$ and the difference between the risk at a candidate $\bar{Q}$ and the risk at the true $\bar{Q}_0$ corresponds with an expected squared error between $\bar{Q}$ and $\bar{Q}_0$ across all values of (A, W):

$$E_0 L(O, \bar{Q}) - E_0 L(O, \bar{Q}_0) = E_0(\bar{Q} - \bar{Q}_0)^2(A, W).$$

Minimizing the expected loss will bring the chosen candidate closer to the true $\bar{Q}_0$ with respect to the dissimilarity measure implied by the loss function, namely, the difference of the risk at $\bar{Q}$ and the optimal risk at $\bar{Q}_0$. How do we find out which algorithm among a library of algorithms yields the smallest expected loss, or, equivalently, which one has the best performance with respect to the dissimilarity implied by the loss function?

3.3 Super (Machine) Learning

Let us return to our simplified mortality study example. The outcome Y is binary, indicating death within 5 years of baseline, and A is also binary, indicating whether the subject meets recommended levels of physical activity. The data structure is $O = (W, A, Y) \sim P_0$. For now let us consider only the covariates $W = \{W_1, W_2, W_3\}$. Age (W_1) is a continuous measure, gender (W_2) is binary, and chronic health history (W_3) is a binary measure indicating whether the subject has a chronic health condition at baseline. While we are ultimately interested in the effect of LTPA on death demonstrated in the next chapter, if we were strictly interested in a *prediction* research question, we could also include LTPA as a covariate in vector W.

Suppose there are three subject matter experts, and they each have a different proposal about the specification of a logistic regression in a parametric statistical model, incorporating their subject matter knowledge. The first believes a main terms statistical model is sufficient for the estimation of the prediction target parameter:

$$\bar{Q}_n^a(A, W) = P_n^a(Y = 1 \mid A, W) = \text{expit}\,(\alpha_{0,n} + \alpha_{1,n}A + \alpha_{2,n}W_1 + \alpha_{3,n}W_2 + \alpha_{4,n}W_3).$$

The second expert proposes including all covariates W and exposure A, as well as an interaction term between age and gender:

$$\bar{Q}_n^b(A, W) = \text{expit}\,(\alpha_{0,n} + \alpha_{1,n}A + \alpha_{2,n}W_1 + \alpha_{3,n}W_2 + \alpha_{4,n}W_3 + \alpha_{5,n}(W_1 \times W_2)).$$

The third expert wants to use a statistical model with main terms and age^2:

$$\bar{Q}_n^c(A, W) = \text{expit}\,(\alpha_{0,n} + \alpha_{1,n}A + \alpha_{2,n}W_1 + \alpha_{3,n}W_2 + \alpha_{4,n}W_3 + \alpha_{5,n}W_1^2).$$

The investigators would ideally like to run all three of these statistical models. Now that we've defined a criterion for the best estimator of $\bar{Q}_0$, how can we responsibly select the optimal estimator from a collection of algorithms, such as the collection of estimators $\bar{Q}_n^a$, $\bar{Q}_n^b$, and $\bar{Q}_n^c$?

3.3.1 Discrete Super Learner

We start by introducing discrete super learning, which will give us an estimate of the cross-validated risk for each algorithm. The entire data set (learning set) is divided into V groups of size $\sim n/V$. These groups are mutually exclusive and exhaustive sets. Our mortality data set has $n = 2066$ subjects. If we want to perform V-fold cross-validation in our discrete super learning procedure, using 10 folds, we will divide our data set into groups of size $\sim 2066/10$. (This gives us four groups with 206 subjects and six groups with 207 subjects.) We label each group from 1 to 10.

Let us focus first on understanding the procedure with just one of the regressions in the collection of algorithms. The observations in group 1 are set aside, and the first regression is fit on the remaining nine groups (called the training set). Then we take the observations in group 1 (called the validation set) and obtain predicted probabilities of death for these 206 or 207 observations using the regression fit on the training set. It is important to note that the observations in group 1 *were not included in the fitting process* and will only be used to evaluate the performance of the predictor that was obtained on the training sample. In this way, we have succeeded in obtaining predicted probabilities of death for approx. 10% of our data, where the prediction function used to obtain these predicted probabilities was fit based on the remaining 90% of data. At this stage, we calculate the estimated risk within the validation set using their predicted probabilities. This procedure is performed for all of the algorithms in the collection of algorithms, so that we have, at the end of the first fold, predicted probabilities for each of the three regressions. We also have an estimate of risk within the validation set (group 1) for each of the three regressions, calculated using their corresponding predicted probabilities.

We need to perform this procedure nine more times, so that each group has the opportunity to take on the role of the validation set and obtain predicted probabilities for each algorithm fit on the corresponding training set. Thus, the procedure continues until we have predicted probabilities of death for all 2066 subjects for each algorithm, and also estimated risk within each validation set for each algorithm. We then have 10 estimated risks for each of the three algorithms, and these risks are averaged across validation sets resulting in one estimated cross-validated risk for

each algorithm. The discrete super learner algorithm selects the algorithm with the smallest cross-validated risk. The algorithm with the smallest cross-validated risk is the "best" estimator according to our criterion: minimizing the estimated expected squared error loss function. See Fig. 3.1 for a diagram of this procedure.

We have now described a new algorithm that took as input the three algorithms. This new estimator is what we call the discrete super learner, and it is indexed by this collection of three algorithms.

By incorporating a rich collection of algorithms that vary in bias and degree of data-fitting, the cross-validation within the discrete super learner prevents overfitting and it also prevents selecting a fit that is too biased. There are many forms of cross-validation, and here we discussed V-fold cross-validation due to its low computational burden while still providing the desirable finite sample and asymptotic optimality properties, which will be discussed later. The collection of algorithms can be large and includes other algorithms besides parametric statistical models, for example, the collection of algorithms may include random forest algorithms and support vector machines.

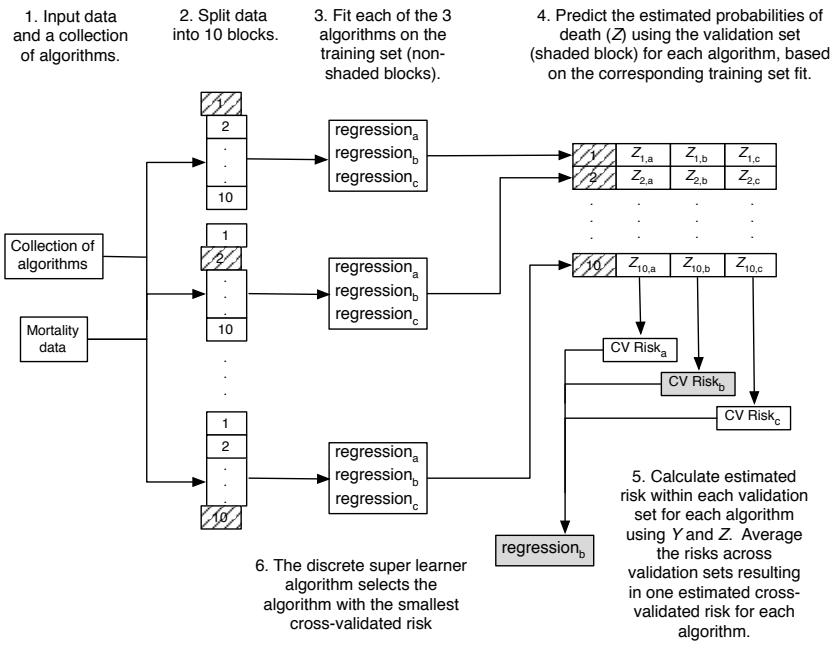

Fig. 3.1 Discrete super learner algorithm for the mortality study example where $\bar{Q}_n^b(A, W)$ is the algorithm with the smallest cross-validated risk

When the loss function is bounded, it has been shown that this discrete super learner will, for large sample sizes, perform as well as the algorithm that is the minimizer of the expected loss function. The latter impossible choice is called the oracle selector, which corresponds with simply selecting the estimator that is closest to the true $\bar{Q}_0$. In addition, cross-validation selection is tailored for small sample sizes, thus one should not be misled that cross-validation requires large sample sizes.

3.3.2 Super Learner

Can we improve upon the discrete super learner? Yes! We can use our three regressions to build a library of algorithms consisting of all weighted averages of these regressions. It is reasonable to expect that one of these weighted averages might perform better than one of the three regressions alone. This simple principle allows us to map a collection of candidate algorithms (in this case, our three regressions) into a library of weighted averages of these algorithms. Each weighted average is a unique candidate algorithm in this augmented library. We can then apply the same cross-validation selector to this augmented set of candidate algorithms, resulting in the super learner. It might seem that the implementation of such an estimator is problematic, since it requires minimizing the cross-validated risk over an infinite set of candidate algorithms (the weighted averages). The contrary is true. The super learner is not more computer intensive than the discrete super learner. If the discrete super learner has been implemented, then all the work has been done! Only the relatively trivial calculation of the optimal weight vector needs to be completed.

Consider that the discrete super learner has already been completed as described in Sect. 3.3.1. We then propose a family of weighted combinations of the three regression algorithms, which we index by the weight vector α. We want to determine which combination minimizes the cross-validated risk over the family of weighted combinations. The (cross-validated) probabilities of death (Z) for each algorithm are used as inputs in a working (statistical) model to predict the outcome Y. Therefore, we have a working model with three $\alpha = \{\alpha_a, \alpha_b, \alpha_c\}$ coefficients that need to be estimated, one for each of the three algorithms. Selecting the weights that minimize the cross-validated risk is a simple minimization problem, formulated as a regression of the outcomes Y on the predicted values of the algorithms (Z) according to the user-supplied parametric family of weighted combinations. The weighted combination with the smallest cross-validated risk is the "best" estimator according to our criterion: minimizing the estimated expected squared error loss function.

The selected weighted combination is a new estimator we can now use to input data (e.g., our complete mortality data set) to estimate predicted probabilities. Thus, we fit each of the three algorithms on our complete data (learning set). Combining these algorithm fits with our new estimator generates the super learner prediction function. This prediction function is the weighted combination of the candidate algorithms applied to the whole data set. See Fig. 3.2 for a full diagram of the super learner algorithm. In order to calculate an honest risk for the super learner, the super

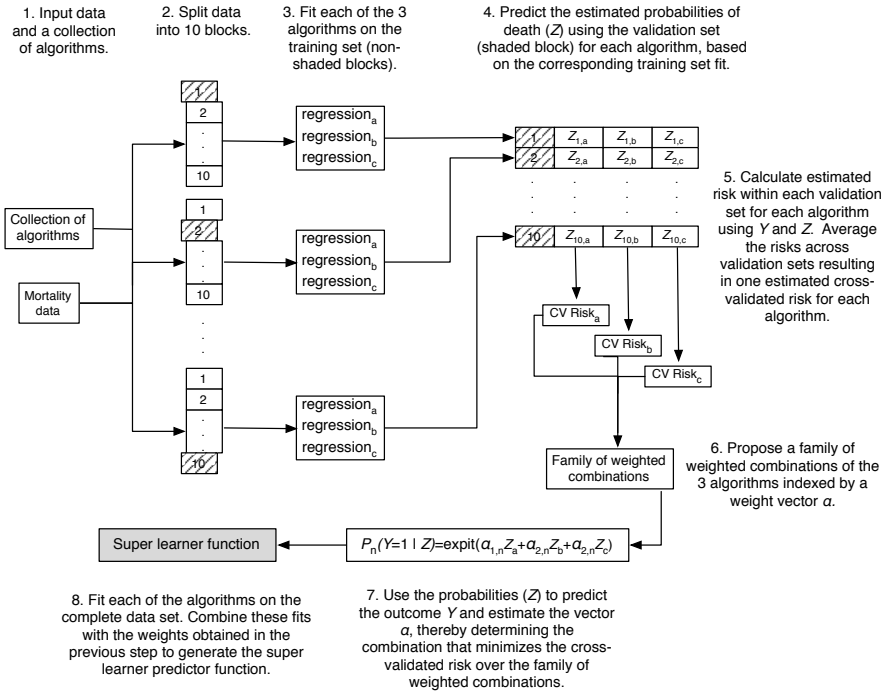

Fig. 3.2 Super learner algorithm for the mortality study example

learner itself must be externally cross-validated after the procedure described above has been implemented.

The family of weighted combinations includes only those α-vectors that have a sum equal to one, and where each weight is positive or zero. Theory does not dictate any restrictions on the family of weighted combinations used for assembling the algorithms; however, the restriction of the parameter space for α to be the convex combination of the algorithms provides greater stability for the final super learner prediction. The convex combination is not only empirically motivated, but also supported by theory. The oracle results for the super learner require a bounded loss function. Restricting oneself to a convex combination of algorithms implies that if each algorithm in the library is bounded, the convex combination will also be bounded.

The super learner improves asymptotically on the discrete super learner by working with a larger library. We reiterate that asymptotic results prove that in realistic scenarios (where none of the algorithms are a correctly specified parametric model), the cross-validated selector performs asymptotically as well as the oracle, which we define as the best estimator given the algorithms in the collection of algorithms. Consequently, the super learner performs asymptotically as well as the best choice among the family of weighted combinations of estimators. Thus, by adding more

competitors, we only improve the performance of the super learner. The asymptotic equivalence remains true if the number of algorithms in the library grows very quickly with sample size. Even when the collection of algorithms contains a correctly specified parametric statistical model, the super learner will approximate the truth as fast as the parametric statistical model, although it will be more variable.

The super learner algorithm provides a system to combine multiple estimators into an improved estimator, and returns a function we can also use for prediction in new data sets.

3.3.3 Finite Sample Performance and Applications

To examine the finite sample performance of the super learner we present a series of simulations and data applications. (For those readers unfamiliar with simulation, simulated data are ideal for methodology validation, as the true underlying distribution of the data is known.) We then demonstrate the super learner on a collection of real data sets and a microarray cancer data set.

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

$$\begin{aligned}
&\textbf{Simulation 1: } Y = -2 \times \mathrm{I}(X < -3) + 2.55 \times \mathrm{I}(X > -2) - 2 \times \mathrm{I}(X > 0) \\
&\qquad\qquad + 4 \times \mathrm{I}(X > 2) - 1 \times \mathrm{I}(X > 3) + U; \\
&\textbf{Simulation 2: } Y = 6 + 0.4X - 0.36X^2 + 0.005X^3 + U; \\
&\textbf{Simulation 3: } Y = 2.83 \times \sin\left(\frac{\pi}{2} \times X\right) + U; \\
&\textbf{Simulation 4: } Y = 4 \times \sin(3\pi \times X) \times \mathrm{I}(X > 0) + U,
\end{aligned}$$

where $\mathrm{I}(\cdot)$ is the usual indicator function and U, our exogenous background error, is drawn from an independent standard normal distribution in all simulations. A sample of size 100 was drawn for each scenario. Figure 3.3 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 regression functions, but all four have the same optimal $R^2 = 0.80$. The empirical R^2 is computed as $R^2 = 1 - (\sum(Y_i - Y_{i,n})^2 / \sum(Y_i - \bar{Y})^2)$, where $\bar{Y} = 1/n \sum_{i=1}^{n} Y_i$ and $Y_{i,n}$ is the predicted value of Y_i reported by the algorithm when applied to the whole data set. The optimal R^2 is the value attained when the true regression function (i.e., true conditional mean) is used and an infinite test sample is used to evaluate the mean squared errors in the numerator and denominator. Knowledge of the true regression function and using an infinite test sample implies $\sum(Y_i - Y_{i,n})^2 = \mathrm{var}(U) \times n = 1 \times n$. Hence the optimal R^2 in all four simulations

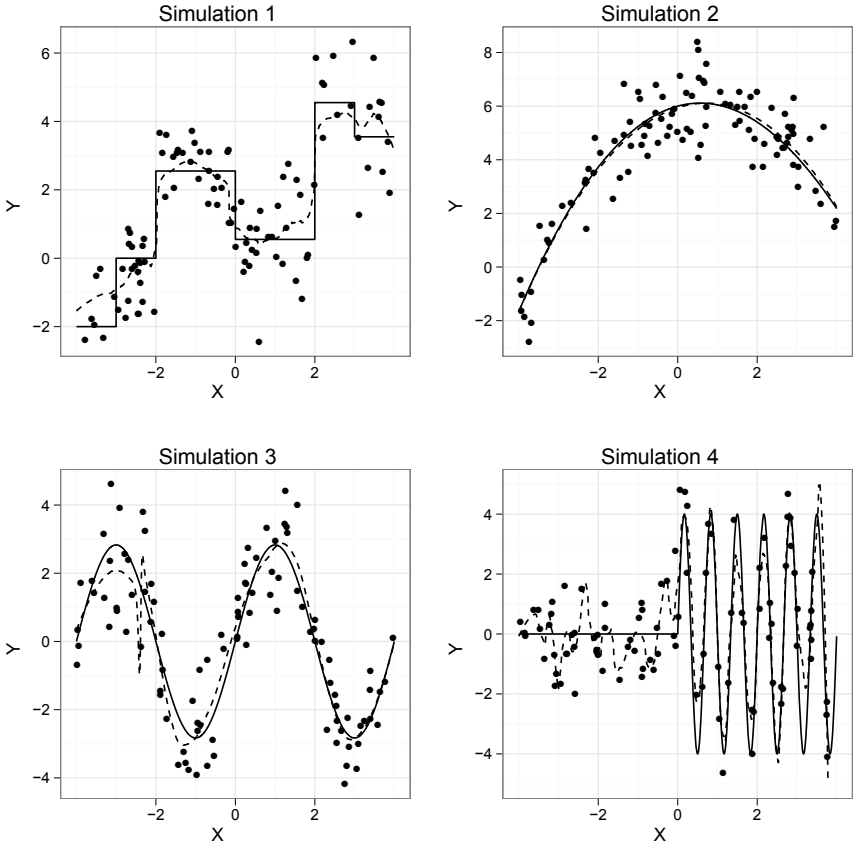

Fig. 3.3 Scatterplots of the four simulations. The *solid line* is the true relationship. The *points* represent one of the simulated data sets of size $n = 100$. The *dashed line* is the super learner fit for the shown data set

is $R^2_{opt} = 1 - (1/\text{var}(Y))$. The variance of Y is set such that $R^2_{opt} = 0.80$ in each simulation.

The collection 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, generalized additive models require a degrees-of-freedom value for the spline functions and 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 collection of algorithms could contain three generalized additive models with degrees of freedom equal to 2, 3, and 4. When one considers different values of tuning parameters as unique prediction algorithms in the collection, it is easy to see how the number of algorithms in the collection can become large.

Table 3.1 Collection of prediction algorithms for the simulations and citations

R Algorithm	Description	Source
glm	Linear model	R Development Core Team (2010)
interaction	Polynomial linear model	R Development Core Team (2010)
randomForest	Random forest	Liaw and Wiener (2002) Breiman (2001b)
bagging	Bootstrap aggregation of trees	Peters and Hothorn (2009) Breiman (1996d)
gam	Generalized additive models	Hastie (1992) Hastie and Tibshirani (1990)
gbm	Gradient boosting	Ridgeway (2007) Friedman (2001)
nnet	Neural network	Venables and Ripley (2002)
polymars	Polynomial spline regression	Kooperberg (2009) Friedman (1991)
bart	Bayesian additive regression trees	Chipman and McCulloch (2009) Chipman et al. (2010)
loess	Local polynomial regression	Cleveland et al. (1992)

In all four simulations, we started with the same collection of 21 prediction algorithms. Table 3.1 contains a list of the algorithms in the library. A linear model and a linear model with a quadratic term were considered. The default random forest algorithm, along with a collection of bagging regression trees with values of the complexity parameter (cp) equal to 0.10, 0.01, and 0.00 and a bagging algorithm adjusting the minimum split parameter to be 5, with default cp of 0.01, was also within the collection of algorithms. Generalized additive models with degrees of freedom equal to 2, 3, and 4 were added along with the default gradient boosting model. Neural networks with sizes 2 through 5, the polymars algorithm, and the Bayesian additive regression trees were added. Finally, we considered the loess curve with spans equal to 0.75, 0.50, 0.25, and 0.10.

Figure 3.3 contains the super learner fit on a single simulated data set for each scenario. With the given collection of algorithms, the super learner is able to adapt to the underlying structure of the data-generating function. For each algorithm we evaluated the true R^2 on a test set of size 10,000. The optimal R^2 is the value attained with knowledge of the true regression function. This value gives us an upper bound on the possible R^2 for each algorithm.

To assess the performance of the super learner in comparison to each algorithm, we simulated 100 samples of size 100 and computed the R^2 for each fit of the true regression function. The results are presented in Table 3.2. Negative R^2 values indicate that the mean is a better predictor of Y than the algorithm. In the first simulation, the regression-tree-based methods perform the best. Bagging complete regression trees (cp = 0) has the largest R^2. In the second simulation, the best algorithm is the quadratic linear regression (SL.interaction). In both of these cases, the super learner is able to adapt to the underlying structure and has an average R^2 close to the best algorithm. 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

Table 3.2 Results for four simulations. Average R^2 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)
Super learner	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

algorithm. Since the individual best algorithm is not known a priori, if a researcher selected a single algorithm, they may do well in some data sets, but the overall performance will be worse than that of the super learner. For example, an individual who always uses bagging complete trees (SL.bagging(0.0)) will do well on the first three simulations, but will perform poorly on the fourth simulation compared to the average performance of the super learner.

In the first three simulations the super learner approaches the optimal R^2 value because algorithms in the collection approximate the truth well. However, in the fourth simulation, the collection is not rich enough to contain a combination of algorithms that approaches the optimal value. The super learner does as well as the best algorithms in the library but does not attain the optimal R^2. Upon supplementation of the collection of algorithms, the super learner achieves an average $R^2 = 0.76$, which is close to the optimal R^2 (results not shown; see Polley and van der Laan 2010).

To study the super learner in real data examples, we collected a number of publicly available data sets. Table 3.3 contains descriptions of the data sets, which can be found either in public repositories or in textbooks, with the corresponding citation listed in the table. 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 collection of prediction algorithms included

Table 3.3 Description of data sets, where n is the sample size and p is the number of covariates

Name	n	p	Source
ais	202	10	Cook and Weisberg (1994)
diamond	308	17	Chu (2001)
cps78	550	18	Berndt (1991)
cps85	534	17	Berndt (1991)
cpu	209	6	Kibler et al. (1989)
FEV	654	4	Rosner (1999)
Pima	392	7	Newman et al. (1998)
laheart	200	10	Afifi and Azen (1979)
mussels	201	3	Cook (1998)
enroll	258	6	Liu and Stengos (1999)
fat	252	14	Penrose et al. (1985)
diabetes	366	15	Harrell (2001)
house	506	13	Newman et al. (1998)

the applicable algorithms from the univariate simulations along with the algorithms listed in Table 3.4. These algorithms represent a diverse set and should allow the super learner to work well in most practical settings. For comparison across data sets, we kept the collection of algorithms fixed for all data analyses.

In order to compare the performance of the K prediction algorithms across diverse data sets with outcomes on different scales, we used the relative mean squared error, which we denote RE for relative efficiency. The denominator is the mean squared error of a linear model:

$$\mathrm{RE}(k) = \frac{\mathrm{MSE}(k)}{\mathrm{MSE}(lm)}, \quad k = 1, \ldots, K.$$

The results for the super learner, the discrete super learner, and each individual algorithm can be found in Fig. 3.4. 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 outperformed the discrete super learner, and both outperformed any individual algorithm. With real data, it is unlikely that one single algorithm would contain the true relationship, and the benefit of the combination of the algorithms vs. the selection of a single algorithm is demonstrated. The additional estimation of the combination parameters (α) does not cause an overfit in terms of the risk assessment. Among the individual algorithms, the Bayesian additive regression trees perform the best, but they overfit one of the data sets with a relative mean squared error of almost 3.0.

A common application of prediction is in microarray data. Super learning is well suited for this setting. Microarray data are often high dimensional, i.e., the number of covariates is larger than the sample size. We demonstrate the super learner in microarray data using a publicly available breast cancer data set published in van't Veer et al. (2002). This study was conducted to develop a gene-expression-based predictor for 5-year distant metastases. The outcome is a binary indicator that a

Table 3.4 Additional prediction algorithms in the collection of algorithms for the real data examples to be combined with the algorithms from Table 3.1

R Algorithm	Description	Source
bayesglm	Bayesian linear model	Gelman et al. (2010) Gelman et al. (2009)
glmnet	Elastic net	Friedman et al. (2010a) Friedman et al. (2010b)
DSA	DSA algorithm	Neugebauer and Bullard (2009) Sinisi and van der Laan (2004)
step	Stepwise regression	Venables and Ripley (2002)
ridge	Ridge regression	Venables and Ripley (2002)
svm	Support vector machine	Dimitriadou et al. (2009) Chang and Lin (2001)

Fig. 3.4 Tenfold cross-validated relative mean squared error compared to glm across 13 real data sets. Sorted by geometric mean, denoted by the plus (+) sign

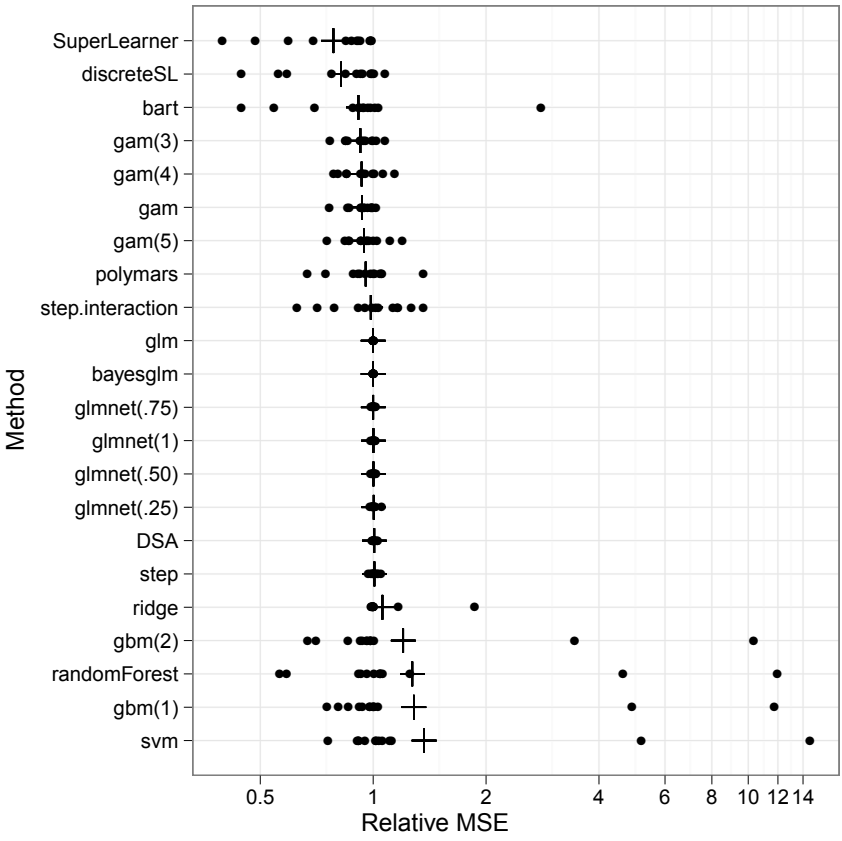

Table 3.5 Twentyfold cross-validated mean squared error for each algorithm and the standard error in the breast cancer study

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)	cor($p < 0.1$)	0.262	0.0232
SL.knn(10)	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)	cor($p < 0.1$)	0.233	0.0168
SL.knn(20)	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)	cor($p < 0.1$)	0.232	0.0139
SL.knn(30)	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)	cor($p < 0.1$)	0.236	0.0118
SL.knn(40)	cor($p < 0.01$)	0.219	0.0151
SL.knn(40)	glmnet	0.211	0.0208
SL.glmnet(1.0)	cor(Rank = 50)	0.229	0.0285
SL.glmnet(1.0)	cor(Rank = 20)	0.208	0.0260
SL.glmnet(0.75)	cor(Rank = 50)	0.221	0.0269
SL.glmnet(0.75)	cor(Rank = 20)	0.209	0.0258
SL.glmnet(0.50)	cor(Rank = 50)	0.226	0.0269
SL.glmnet(0.50)	cor(Rank = 20)	0.211	0.0256
SL.glmnet(0.25)	cor(Rank = 50)	0.230	0.0266
SL.glmnet(0.25)	cor(Rank = 20)	0.216	0.0252
SL.randomForest	Clinical	0.198	0.0186
SL.randomForest	cor($p < 0.01$)	0.204	0.0179
SL.randomForest	glmnet	0.220	0.0245
SL.bagging	Clinical	0.207	0.0160
SL.bagging	cor($p < 0.01$)	0.205	0.0184
SL.bagging	glmnet	0.206	0.0219
SL.bart	Clinical	0.202	0.0183
SL.bart	cor($p < 0.01$)	0.210	0.0207
SL.bart	glmnet	0.220	0.0275
SL.mean	All	0.224	0.1016

patient had a distant metastasis within 5 years of initial therapy. In addition to the expression data, six clinical variables were attained. The clinical information was age, tumor grade, tumor size, estrogen receptor status, progesterone receptor status, and angioinvasion. The array data contained 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.

In high-dimensional data, it is often beneficial to screen the variables before running prediction algorithms. Screening is part of the algorithm and should thus also be included when calculating the cross-validated risk of an algorithm in the super learner. 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 k-nearest neighbors on the subset of only clinical variables. These two algorithms are considered unique algorithms. Another screening algorithm involves testing the pairwise correlations of each variable with the outcome and ranking the variables by the corresponding p-value. With the ranked list of variables, we consider the screening cutoffs as follows: 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 involves running the glmnet algorithm and selecting the variables with nonzero coefficients.

The results for the breast cancer data can be found in Table 3.5. The algorithms in the collection 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 this collection 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 was able to attain a risk comparable to the best algorithm (MSE = 0.194).

3.4 Road Maps

In previous chapters, we introduced our road map for targeted learning (Fig. 3.5), the first steps of which involved defining our data, model, and target parameter. This chapter dealt with obtaining the best initial estimator of the relevant portion Q_0 of the distribution P_0 of O. The next stage of the road map addresses estimation of the target parameter using TMLE, taking this initial estimator as input.

We also present a separate road map for prediction estimation questions in Fig. 3.6. We note that inference for prediction is not covered in detail in this text and we refer readers in Sect. 3.7 to literature using the permutation distribution for obtaining exact tests of the null hypothesis of independence of the covariates and the outcome, allowing the incorporation of machine learning.

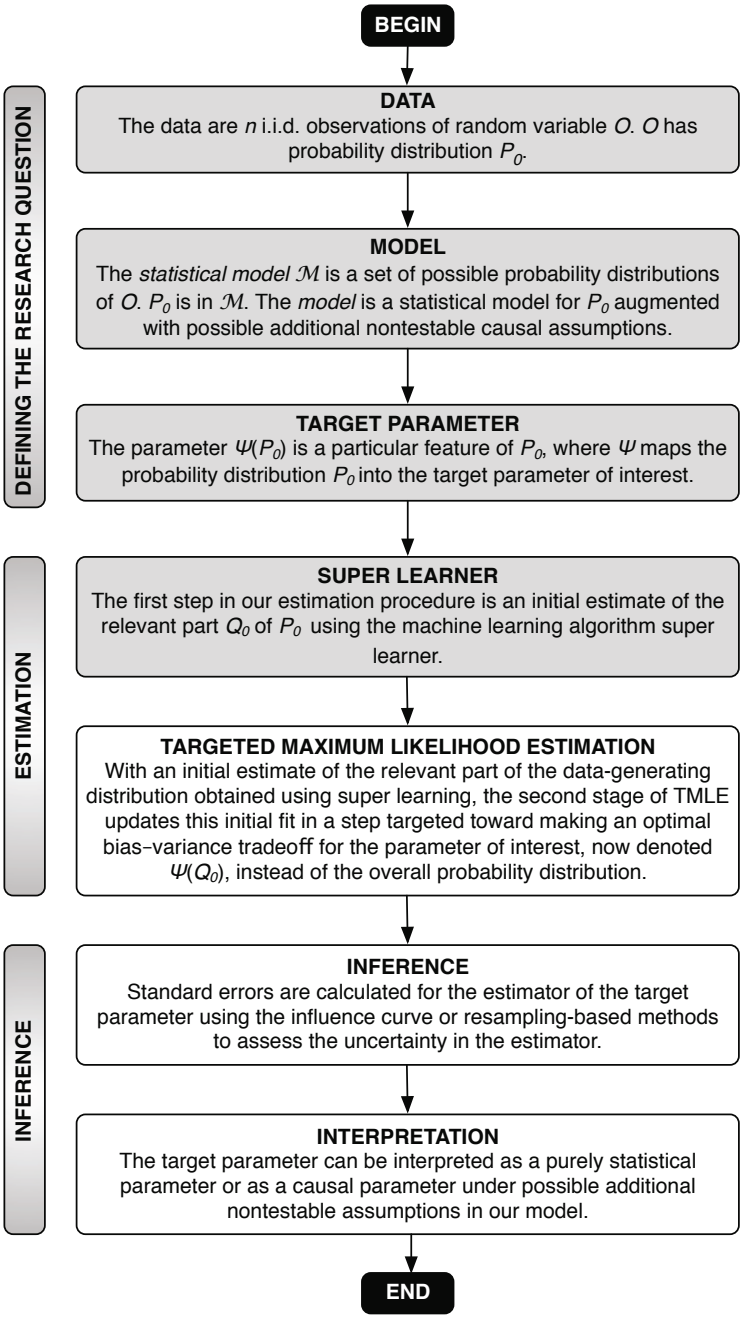

Fig. 3.5 Road map for targeted learning

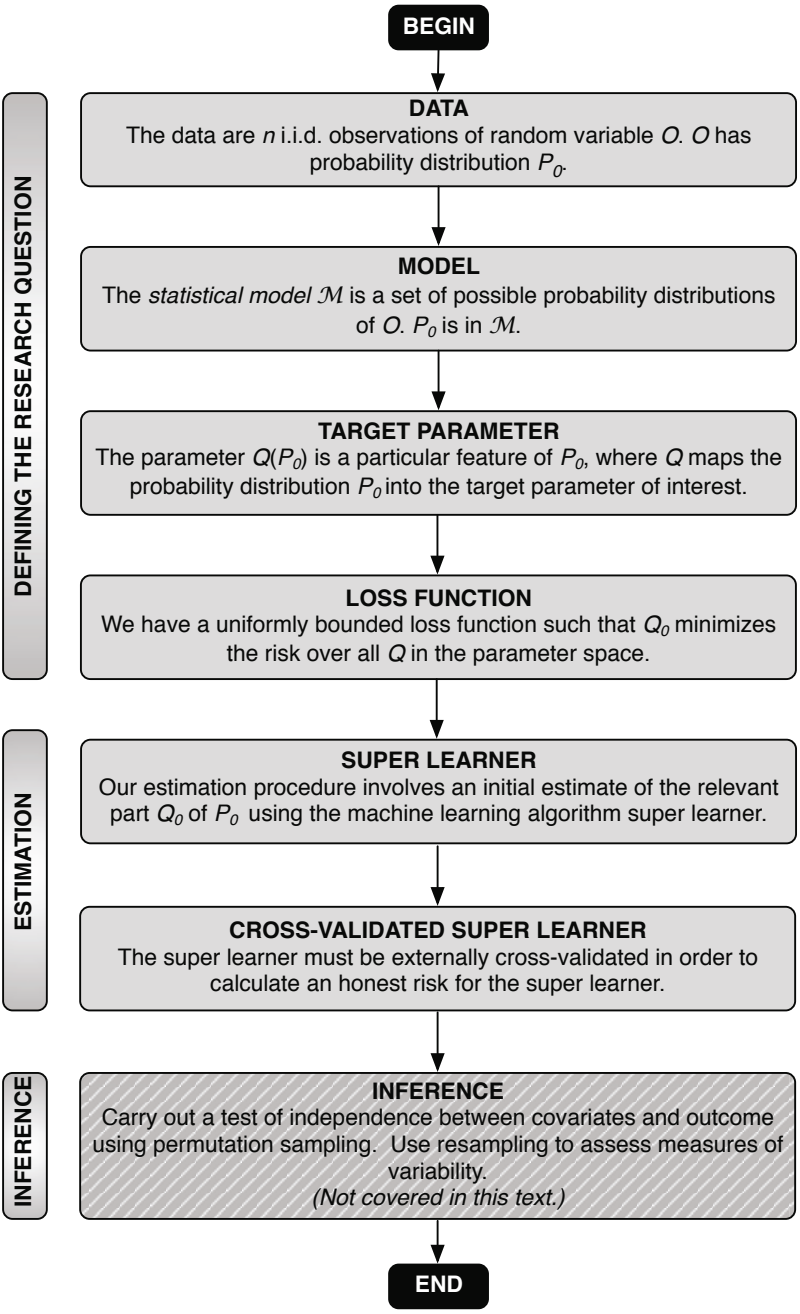

Fig. 3.6 Road map for prediction

3.5 Conceptual Framework of Loss-Based Super Learning

Suppose we observe n i.i.d. observations $O_1, \ldots, O_n$ on a random variable O from a probability distribution P_0 known to be an element of a statistical semiparametric statistical model $\mathcal{M}$. Our goal is to learn a particular parameter of P_0, which we will denote by $Q(P_0)$, and let $\mathcal{Q} = \{Q(P) : P \in \mathcal{M}\}$ be the parameter space. We assume that we have available a loss function $L(Q)(O)$ such that Q_0 minimizes the risk $P_0 L(Q) \equiv E_0 L(Q)(O)$ of Q over all Q in the parameter space $\mathcal{Q}$. In addition, it is assumed that this loss function is uniformly bounded so that $P_0(L(Q)(O) < M) = 1$ for some universal constant M, uniformly in all $Q \in \mathcal{Q}$. A library of candidate estimators of Q_0, the choice of loss function, and a choice of cross-validation scheme now define the super learner of Q_0.

Creating a better estimator from among the available estimators. If the parameter $Q : \mathcal{M} \to \mathcal{Q}$ is not pathwise differentiable (i.e., not identifiable and smooth enough to allow central-limit-theorem-based inference), then there is no efficiency theory. That is, even asymptotically, there is no best estimator of Q_0. As a consequence, the best one can do is to make sure that one is better than any competitor. This can be done by including any competing algorithm in the super learner collection of algorithms. Thus one has a large collection of candidate estimators. These candidate estimators should use the knowledge that $P_0 \in \mathcal{M}$. That is, each estimator should at a minimum map the data into functions in the parameter space $\mathcal{Q}$.

One particular approach might require a choice of a number of fine-tuning parameters. Such an approach would generate many members for the collection. An estimator could be combined with different dimension-reduction approaches, so that one estimator would result in several members in the collection. One might also partition the outcome space for O and stratify estimators accordingly, and also consider applying different estimators to different strata. In this manner, one estimation procedure and several stratification variables would map into a whole collection of estimators for the super learner library.

There is no point in painstakingly trying to decide which estimators to enter in the collection; instead add them all. The theory supports this approach, and finite sample simulations and data analyses only confirm that it is very hard to overfit the super learner by augmenting the collection, but benefits are obtained. Indeed, for large data sets, we simply do not have enough algorithms available to build the desired collection that would fully utilize the power of the super learning principle as established by the oracle result.

The free lunch: fully robust application-specific modeling. This is not enough. In a particular application, there are always many experts with creative ideas. Put them in a room and let them generate ideas about effective dimension reductions, propose parametric statistical models they find interesting, and let them propose strategies for approximating this unknown true target-function Q_0. Translate these into new candidate algorithms for the collection of algorithms. For example, one professor might think that certain specific summary measures of the history of the unit at

baseline should be particularly effective in predicting the outcome of interest. This then translates into estimators that only use these summaries and algorithms that add these summary measures to the existing set of (nontransformed) variables. If there are competing theories about how the truth is best approximated, translate them all into candidate estimators for the super learning library. The super learner will not select an estimator that performs poorly, but even mediocre algorithms can still improve the super learner. That is, there is no risk in adding candidate estimators that are heavily model-based to the super learner; there is only benefit.

Concerned about overfitting the super learner? Indeed, let the data speak to answer this question. That is why one should evaluate the performance of the super learner itself by determining its cross-validated risk. It can then be determined if the super learner does as well or better than any of the candidate algorithms in the collection. In particular, one might diagnose that one has reached a point at which adding more algorithms harms the super learner performance, but our experience has not reached that point by any meaningful standard. If anything, it appears to flatten out, but not deteriorate. However, it is important that one use quite high-fold cross-validation when evaluating the super learner itself (say 20-fold), especially when the sample size gets small. Above all, it is crucial that the family of combinations respects a universal bound on the loss functions across all combinations.

Computational challenge. The super learning system of learning is perfectly tailored for parallel programming. The different candidate estimators can do their job separately, and the applications of the candidate estimators to the different training sets can be separated as well.

Generality of super learning. Super learning can be applied to estimate an immense class of parameters across different data structures O and different statistical models $\mathcal{M}$. One can use it to estimate marginal densities, conditional densities, conditional hazards, conditional means, conditional medians, conditional quantiles, conditional survival functions, and so on, under biased sampling, missingness, and censoring.

It is a matter of defining Q_0 as a parameter of P_0 and determining an appropriate loss function. For example, the minus log loss function can be used for conditional densities and hazards. However, one might come up with loss functions that are indexed by unknown parameters: $L_h(Q)$ for some unknown h. Many such examples are now provided in the literature, such as the double robust augmented inverse probability of treatment-weighted loss function for the treatment-specific mean outcome as a function of effect modifiers. In this case h includes a conditional distribution of treatment as a function of the covariates. This nuisance parameter would be known in an RCT, but it will need to be estimated in an observational study. We discuss the statistical property of double robustness in Chap. 6.

In these situations one will need to estimate h from the data and then employ this estimated loss function as before. The basic messsage is that one needs the estimation of h to be easier than the estimation of Q_0 in order to get the full benefit

of super learner, as if h was known from the start. It should also be remarked that h could represent an index that does not affect the validity of the loss function in the sense that for each choice of h, Q_0 minimizes the risk of $L_h(Q)$ over all Q. In these cases, the choice of h only affects the dissimilarity measure for which the performance of the super learner is optimized. For example, h might represent a weight function in a squared-error loss function, or it might represent a covariance matrix for the generalized squared-error loss function for a conditional mean $E(\mathbf{Y} \mid W)$ of a multivariate outcome:

$$L_h(Q)(W, \mathbf{Y}) = (\mathbf{Y} - Q(W))^\top h(W)(\mathbf{Y} - Q(W)).$$

Choice of loss function. If several choices are available, the loss function that maps into the desired dissimilarity measure $d_L(Q, Q_0) = E_0 L(Q) - E_0 L(Q_0)$ should be selected. It should be kept in mind that the super learner optimizes the approximation of Q_0 with respect to this dissimilarity d_L implied by the loss function. For example, suppose one wishes to estimate a conditional survival function at a time point t_0, $Q_0 = P(T > t_0 \mid W)$. Then one could still use the minus log loss function for the conditional density of T, given W, which, by substitution, also implies a valid loss function for Q_0, since Q_0 is determined by this conditional density. However, this loss function is trying to determine an entire conditional density, and is thus not very targeted towards its goal. Instead, we can use

$$L(Q)(W, T) = [I(T > t_0) - Q(W)]^2 .$$

This loss function is minimized over all functions Q of W by $Q_0 = P(T > t_0 \mid W)$, and thereby targets exactly our parameter of interest. Indeed, the super learner will now be aiming to minimize the dissimilarity:

$$d_L(Q, Q_0) = E_0[Q(W) - Q_0(W)]^2,$$

i.e., the expected squared error between the candidate survival function at t_0 and the true survival function at t_0.

Formal oracle result for cross-validation selector. Consider a loss function that satisfies

$$\sup_Q \frac{\mathrm{var}_{P_0}\{L(Q) - L(Q_0)\}}{P_0\{L(Q) - L(Q_0)\}} \leq M_2 \tag{3.1}$$

and that is uniformly bounded:

$$\sup_{O,Q} \mid L(Q) - L(Q_0) \mid (O) < M_1 < \infty,$$

where the supremum is over the support of P_0 and over all possible candidate estimators of Q_0 that will ever be considered. We used the notation $P_0 f = \int f(o) dP_0(o)$ for the expectation of $f(O)$ under P_0. The first property (3.1) applies to the log-likelihood loss function and any weighted squared residual loss function, among

others. Property (3.1) is essentially equivalent to the assumption that the loss-function-based dissimilarity $d(Q, Q_0) = P_0\{L(Q) - L(Q_0)\}$ is quadratic in a distance between Q and Q_0. Property (3.1) has been proven for log-likelihood loss functions and weighted L^2-loss functions and is in essence equivalent to stating that the loss function implies a quadratic dissimilarity $d(Q, Q_0)$ (van der Laan and Dudoit 2003). If this property does not hold for the loss function, the rates $1/n$ for second-order terms in the below stated oracle inequality reduce to the rate $1/\sqrt{n}$.

Let $B_n \in \{0, 1\}^n$ be a random variable that splits the learning sample in a training sample $\{i : B_n(i) = 0\}$ and validation sample $\{i : B_n(i) = 1\}$, and let P^0_{n,B_n} and P^1_{n,B_n} denote the empirical distribution of the training and validation sample, respectively. Given candidate estimators $P_n \to \hat{Q}_k(P_n)$, the loss-function-based cross-validation selector is now defined by

$$k_n = \hat{K}(P_n) = \arg\min_k E_{B_n} P^1_{n,B_n} L(\hat{Q}_k(P^0_{n,B_n})).$$

The resulting estimator, the discrete super learner, is given by $\hat{Q}(P_n) = \hat{Q}_{\hat{K}(P_n)}(P_n)$.

For quadratic loss functions, the cross-validation selector satisfies the following (so-called) oracle inequality: for any $\delta > 0$

$$E_{B_n}\{P_0 L(\hat{Q}_{k_n}(P^0_{n,B_n}) - L(Q_0)\} \le (1 + 2\delta) E_{B_n} \min_k P_0\{L(\hat{Q}_k(P^0_{n,B_n})) - L(Q_0)\} + 2C(M_1, M_2, \delta)\frac{1 + \log K(n)}{np},$$

where the constant $C(M_1, M_2, \delta) = 2(1+\delta)^2(M_1/3+M_2/3)$ (van der Laan and Dudoit 2003, p. 25). This result proves [see van der Laan and Dudoit (2003) for the precise statement of these implications] that if the number of candidates $K(n)$ is polynomial in sample size, then the cross-validation selector is either asymptotically equivalent to the oracle selector (based on a sample of training sample sizes, as defined on the right-hand side of the above inequality), or it achieves the parametric rate $\log n/n$ for convergence with respect to $d(Q, Q_0) \equiv P_0\{L(Q) - L(Q_0)\}$.

So in most realistic scenarios, in which none of the candidate estimators achieves the rate of convergence one would have with an a priori correctly specified parametric statistical model, the cross-validated estimator selector performs asymptotically exactly as well (not only in rate, but also up to the constant!) as the oracle-selected estimator. These oracle results are generalized for estimated loss functions $L_n(Q)$ that approximate a fixed loss function $L(Q)$. If $\arg\min_Q P_0 L_n(Q) \neq Q_0$, then the oracle inequality also presents second-order terms due to the estimation of the loss function (van der Laan and Dudoit 2003).

3.6 Notes and Further Reading

We've discussed in this chapter the notion of estimator selection. We use this terminology over "model selection," since the formal meaning of a (statistical) model in

the field of statistics is the set of possible probability distributions, and most algorithms are not indexed by a statistical model choice. The general loss-based super learner was initially presented in van der Laan et al. (2007b). Super learner is a generalization of the stacking algorithm introduced in the neural networks context by Wolpert (1992) and adapted to the regression context by Breiman (1996c), and its name was introduced due to the theoretical oracle property and its consequences as presented in van der Laan and Dudoit (2003). The stacking algorithm is examined in LeBlanc and Tibshirani (1996) and the relationship to the model-mix algorithm of Stone (1974) and the predictive sample-reuse method of Geisser (1975) is discussed. Recent literature on aggregation and ensemble learners includes Tsybakov (2003), Juditsky et al. (2005), Bunea et al. (2006, 2007a,b), and Dalalyan and Tsybakov (2007, 2008). As noted previously, inference for prediction, such as permutation resampling, is not covered in this text. We refer the interested reader to Lehmann (1986), Hastie et al. (2001), Ruczinski et al. (2002), Birkner et al. (2005), and Chaffee et al. (2010). The simulations and data analyses contained in this chapter were previously published as a technical report (Polley and van der Laan 2010).

Chapter 15 uses super learning to estimate the risk score of mortality in a Kaiser Permanente database. Additionally, Chap. 16 discusses the use of super learning in right-censored data. We refer readers to Polley and van der Laan (2009) for a chapter in the book *Design and Analysis of Clinical Trials with Time-to-Event Endpoints* that discusses the use of super learning to assess effect modification in clinical trials.

Theory for loss-function-based cross-validation is presented in van der Laan and Dudoit (2003), including the finite sample oracle inequality, the asymptotic equivalence of the cross-validation selector, and the oracle selector. See also van der Laan et al. (2006), van der Vaart et al. (2006), van der Laan et al. (2004), Dudoit and van der Laan (2005), Keleş et al. (2002), and Sinisi and van der Laan (2004). A finite sample result for the single-split cross-validation selector for the squared error loss function was established in Györfi et al. (2002) and then generalized in van der Laan and Dudoit (2003) and Dudoit and van der Laan (2005) for both general cross-validation schemes and a general class of loss functions.

Other types of cross-validation beyond V-fold cross-validation include bootstrap cross-validation, Monte Carlo cross-validation, and leave-one-out cross-validation (Stone 1974, 1977; Breiman et al. 1984; Breiman and Spector 1992; Efron and Tibshirani 1993; Breiman 1996a,b; Ripley 1996; Breiman 1998; Hastie et al. 2001; Ambroise and McLachlan 2002; Györfi et al. 2002). Simulation studies (Pavlic and van der Laan 2003) show that likelihood-based cross-validation performs well when compared to common validity-functionals-based approaches, such as Akaike's information criterion (Akaike 1973; Bozdogan 2000), Bayesian Information criterion (Schwartz 1978), minimum description length (Rissanen 1978), and informational complexity (Bozdogan 1993).

Hastie et al. (2001) covers a variety of machine learning algorithms and related topics. Areas include stepwise selection procedures, ridge regression, LASSO, principal component regression, least angle regression, nearest neighbor methods, random forests, support vector machines, neural networks, classification methods, kernel smoothing methods, and ensemble learning.