Modeling the Exposure and the Outcome: Introduction to the Super Learner

Ashley I Naimi

June 2022

Contents

1 Introduction to Stacking

In the early 1990s, Wolpert developed an approach to combine several "lower-level" machine learning methods into a "higher-level" model with the goal of increasing predictive accuracy (Wolpert, 1992). He termed the approach "stacked generalization," or "stacking." Later, Breiman demonstrated how stacking can be used to improve the predictive accuracy in a regression context, and showed that imposing certain constraints on the higher-level model improved predictive performance (Breiman, 1996). More recently, van der Laan and colleagues proved that stacking possesses certain ideal theoretical properties (van der Laan and S, 2003, van der Laan et al. (24), van der Laan et al. (2007)). In particular, their oracle inequality guarantees that in large samples the algorithm will perform at least as well as the best individual predictor included in the ensemble. Therefore, choosing a large library of diverse algorithms will enhance performance, and creating the best weighted combination of candidate algorithms will further improve performance. Here, "best" is defined in terms of a bounded loss function, and over-fitting is avoided with V-fold cross-validation. In this context, the term "Super Learner" was coined.[1]

[1] In other words, super learner is stacking or a stacked generalization.

Super Learner has tremendous potential for improving the quality of prediction algorithms in applied health sciences, and minimizing the extent to which empirical findings rely on parametric modeling assumptions. It can also serve to be a tremendously important tool when seeking to estimate causal effects using machine learning algorithms. Indeed, as we will see there are several software implementations of the super learner that are meant to be merged with various double robust estimators, including AIPW and TMLE.

2 Estimating a Dose-Response Curve with Super Learner

For 1,000 observations, we generate a continuous exposure X by drawing from a uniform distribution with a minimum of zero and a maximum of eight and then generate a continuous outcome Y as

$$Y = 5 + 4 \times \sqrt{9X} \times \mathbb{I}(X < 2) + \mathbb{I}(X \geq 2) \times (|x - 6|^2) + \epsilon, \qquad (1)$$

where $\mathbb{I}()$ denotes the indicator function evaluating to 1 if the argument is true (zero otherwise), and ϵ was drawn from a doubly-exponential distribution with

mean zero and scale parameter one. The true dose-response curve is depicted by the black line in Figure 1.

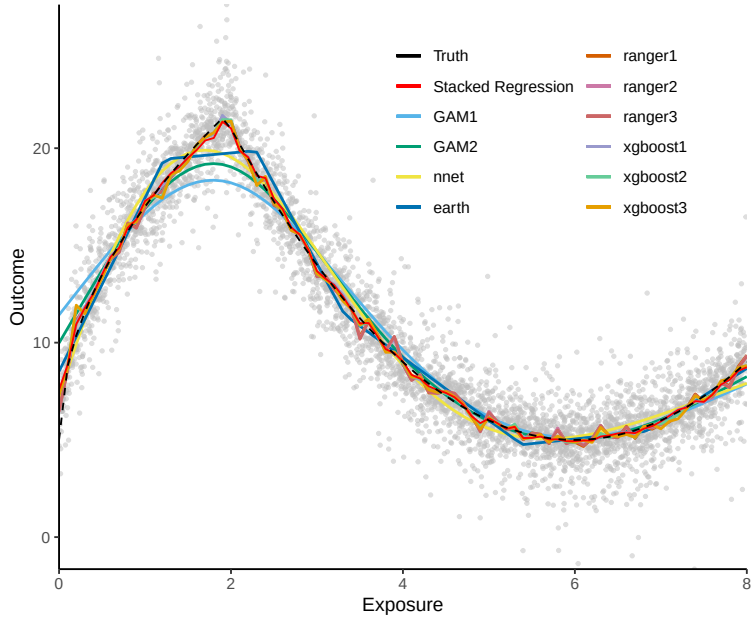

Figure 1: Example dose-response relation between a continuous variable X and an outcome variable Y. The black dashed line represents the true dose-response relation, with all additional lines representing various machine learning estimators of the fit, including the Super Learner (Stacked Regression).

The code to generate these data is as follows:

```
library(rmutil)
library(here)
library(SuperLearner)

# set the seed for reproducibility
set.seed(123)

# generate the observed data
n = 1000
x = runif(n, 0, 8)
y = 5 + 4 * sqrt(9 * x) * as.numeric(x <
    2) + as.numeric(x >= 2) * (abs(x - 6)^(2)) +
    rlaplace(n)

# to plot the true dose-response curve,
# generate sequence of 'doses' from 0
```

```
# to 8 at every 0.1, \tthen generate
# the true outcome
xl <- seq(0, 8, 0.1)
yl <- 5 + 4 * sqrt(9 * xl) * as.numeric(xl <
    2) + as.numeric(xl >= 2) * (abs(xl -
    6)^(2))

D <- data.frame(x, y)  # observed data
Dl <- data.frame(xl, yl)  # for plotting the true dose-response curve

head(D)
```

```
##          x         y
## 1 2.300620 18.082553
## 2 6.306441  5.301833
## 3 3.271815 11.304711
## 4 7.064139  7.359499
## 5 7.523738  8.510791
## 6 0.364452 12.199149
```

We now manually demonstrate how the Super Learner can be used to flexibly model this relation without making parametric assumptions.

To simplify our illustration, we consider only two "level-zero" algorithms as candidates to the Super Learner library: generalized additive models with 5-knot natural cubic splines (`gam`) (Hastie and Tibshirani, 1990) and multivariate adaptive regression splines implemented via the `earth` package. In practice, a large and diverse library of candidate estimators is recommended. Our specific interest is in quantifying the mean of Y as a (flexible) function of X. To measure the performance of candidate algorithms and to construct the weighted combination of algorithms, we select the L-2 squared error loss function $(Y - \hat{Y})^2$ where $\hat{Y}$ denotes our predictions. Minimizing the expectation of the L-2 loss function is equivalent to minimizing mean squared error, which is the same objective function used in ordinary least squares regression (Hastie et al., 2009)[(section 2.4)]. To estimate this expected loss, called the "risk", we use V-fold cross-validation with $V = 5$ folds.

```
# Specify the number of folds for
# V-fold cross-validation
folds = 5
## split data into 5 groups for 5-fold
## cross-validation we do this here so
## that the exact same folds will be
## used in both the SL fit with the R
## package, and the hand coded SL
index <- split(1:1000, 1:folds)
splt <- lapply(1:folds, function(ind) D[index[[ind]],
    ])
# view the first 6 observations in the
# first [[1]] and second [[2]] folds
head(splt[[1]])
```

```
##           x         y
## 1  2.300620 18.082553
## 6  0.364452 12.199149
## 11 7.654667  7.659093
## 16 7.198600  7.229326
## 21 7.116315  6.165418
## 26 5.668244  4.659275
```

```
head(splt[[2]])
```

```
##           x         y
## 2  6.306441  5.301833
## 7  4.224844  8.943891
## 12 3.626673 10.249762
## 17 1.968702 21.964325
## 22 5.542427  6.089999
## 27 4.352528  7.891969
```

```
#-----------------------------------------------------------------------------
# Fit using the SuperLearner Package
#-----------------------------------------------------------------------------
# Create the 5 df GAMs using functions
# called from programs 'sourced' above
SL.gam.5 <- create.Learner("SL.gam", params = list(deg.gam = 5))

# Specifying the SuperLearner library
# of candidate algorithms
sl.lib <- c(SL.gam.5$names, "SL.earth")

# Fit using the SuperLearner package,
# specify \t\toutcome-for-prediction
# (y), the predictors (x), the loss
# function (L2), \t\tthe library
# (sl.lib), and number of folds
fitY <- SuperLearner(Y = y, X = data.frame(x),
    method = "method.NNLS", SL.library = sl.lib,
    cvControl = list(V = folds, validRows = index))

# View the output: 'Risk' column
# returns the CV-MSE estimates
# \t\t'Coef' column gives the weights
# for the final SuperLearner
# (meta-learner)
fitY
```

```
##
## Call:
## SuperLearner(Y = y, X = data.frame(x), SL.library = sl.lib, method = "method.NNLS",
##     cvControl = list(V = folds, validRows = index))
##
##
##                   Risk      Coef
## SL.gam_1_All 2.446902 0.3379264
```

```
## SL.earth_All 2.309112 0.6620736
```

```
# Now predict the outcome for all
# possible x 'doses'
yS <- predict(fitY, newdata = data.frame(x = xl),
    onlySL = T)$pred

# Create a dataframe of all x 'doses'
# and predicted SL responses
Dl1 <- data.frame(xl, yS)
```

Step 1. Split the observed "level-zero" data into 5 mutually exclusive and exhaustive groups of $n/V = 1000/5 = 200$ observations. These groups are called "folds."

Step 2. For each fold $v = \{1, \ldots, 5\}$,

a. Define the observations in fold v as the validation set, and all remaining observations (80% of the data) as the training set.

b. Fit each algorithm on the training set.

c. For each algorithm, use its estimated fit to predict the outcome for each observation in the validation set. Recall the observations in the validation set are not used train each candidate algorithm.

d. For each algorithm, estimate the risk. For the L-2 loss, we average the squared difference between the outcome Y and its prediction $\hat{Y}$ for all observations in the validation set v. In other words, we calculate the mean squared error (MSE) between the observed outcomes in the validation set and the predicted outcomes based on the algorithms fit on the training set.

Step 3. Average the estimated risks across the folds to obtain one measure of performance for each algorithm. In our simple example, the cross-validated estimates of the squared prediction error are 2.45 for `gam` and 2.31 for `earth`.

At this point, we could simply select the algorithm with smallest cross-validated risk estimate (here, `earth`). This approach is sometimes called

the Discrete Super Learner. Instead, we combine the cross-validated predictions, which are referred to as the "level-one" data, to improve performance and build the "level-one" learner.

Step 4. Let $\hat{Y}_{\text{gam-cv}}$ and $\hat{Y}_{\texttt{earth-cv}}$ denote the cross-validated predicted outcomes from `gam` and `earth`, respectively. Recall the observed outcome is denoted Y. To calculate the contribution of each candidate algorithm to the final Super Learner prediction, we use non-negative least squares to regress the actual outcome against the predictions, while suppressing the intercept and constraining the coefficients to be non-negative and sum to 1:

$$\mathbb{E}(Y|\hat{Y}_{\text{gam-cv}}, \hat{Y}_{\text{earth-cv}}) = \alpha_1 \hat{Y}_{\text{gam-cv}} + \alpha_2 \hat{Y}_{\text{earth-cv}}, \tag{2}$$

such that $\alpha_1 \geq 0$; $\alpha_2 \geq 0$, and $\sum_{k=1}^{2} \alpha_k = 1$. Combining the $\hat{Y}_{\text{gam-cv}}$ and $\hat{Y}_{\text{earth-cv}}$ under these constraints (non-negative estimates that sum to 1) is referred to as a "convex combination," and is motivated by both theoretical results and improved stability in practice [@Breiman1996, @vanderLaan2007]. Non-negative least squares corresponds to minimizing the mean squared error, which is our chosen loss function (and thus, fulfills our objective). We then normalize the coefficients from this regression to sum to 1. In our simple example, the normalized coefficient values are $\hat{\alpha}_1$ =0.34 for `gam` and $\hat{\alpha}_2$ =0.66 for `earth`. Therefore, both generalized additive models and regression splines each contribute approximately 35% and 65% of the weight in the optimal predictor.

Step 5. The final step is to use the above weights to generate the Super Learner, which can then be applied to new data (X) to predict the continuous outcome. To do so, re-fit `gam` and `earth` to the entire sample and denote the predicted outcomes as $\hat{Y}_{\text{gam}}$ and $\hat{Y}_{\text{earth}}$, respectively. Then combine these predictions with the estimated weights from Step 4:

$$\hat{Y}_{\text{SL}} = 0.34\hat{Y}_{\text{gam}} + 0.66\hat{Y}_{\text{earth}} \tag{3}$$

where $\hat{Y}_{\text{SL}}$ denotes our final Super Learner predicted outcome.

Figure 1 demonstrates the fit from several different lower-level algorithms (GAM, nnet, earth, ranger, xgboost), and the combined super learner algorithm (Stacked Regression). The code for this Figure is provided in the

supplementary material.

References

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

Trevor Hastie and Robert Tibshirani. *Generalized Additive Models*. Chapman & Hall, London; New York, 1990.

Trevor Hastie, Robert Tibshirani, and Jerome H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, New York, NY, 2009.

Mark J van der Laan and Dudoit S. Unified cross-validation methodology for selection among estimators and a general cross-validated adaptive epsilon-net estimator: Finite sample oracle inequalities and examples. *U.C. Berkeley Division of Biostatistics Working Paper Series*, Working Paper 130 (https://biostats.bepress.com/ucbbiostat/paper130), 2003.

Mark J van der Laan, Eric C Polley, and Alan E Hubbard. Super learner. *Statistical Applications in Genetics and Molecular Biology*, 6(1):Article 25, 2007.

Mark J. van der Laan, Sandrine Dudoit, and Aad W. van der Vaart. The cross-validated adaptive epsilon-net estimator. *Statistics & Decisions*, 373-395, 24.

DH Wolpert. Stacked generalization. *Neural Netw*, 5(2):241–59, 1992.