An Introduction to SuperLearner, a loss-based ensemble prediction algorithm

Scott Grey, PhD

April 14, 2016

Overview

1. Background on the development of super learner and targeted learning
2. Theory behind the super learner algorithm
3. Application of super learner in R
4. Some (hopefully) helpful comments on utilizing super learner

This presentation, the data (with documentation) and R code is available at: https://github.com/sfgrey/Super-Learner-Presentation.git

Background

Background

"Essentially, all models are wrong, but some are useful"
- George Box, 1979

Mantra of statisticians regarding the development of statistical models for many years

In the 1990s an awareness developed among statisticians (Breiman, Harrell) that this approach was wrong

- Parametric model assumptions rarely met
- Large number of variables makes it difficult to correctly specify a model

Simultaneously, computer scientists and some statisticians developed the machine learning field to address the limitations of parametric models

Targeted learning

Combines advanced machine learning with efficient semiparametric estimation to provide a framework for answering causal questions from data

- Developed by Mark van der Laan research group at UC Berkeley
- Started with the seminal 2006 article on targeted maximum likelihood estimation

Central motivation is the belief that statisticians treat estimation as *Art* not **Science**

- This results in misspecified models that are data-adaptively selected, but this part of the estimation procedure is not accounted for in the variance

Estimation is a Science, *Not an Art*

Specific definitions required

1. **Data**: realizations of random variables with a probability distribution
2. **Model**: actual knowledge about the data generating probability distribution
3. **Target Parameter**: a feature of the data generating probability distribution
4. **Estimator**: an a priori-specified algorithm, benchmarked by a dissimilarity-measure (e.g., MSE) w.r.t. target parameter

Theory

Data

Random variable O, observed n times, defined in a simple case as $O = (A, W, Y) \sim P_0$ if we are without common issues such as missingness and censoring

- A: exposure or treatment
- W: vector of covariates
- Y: outcome
- P_0: the true probability distribution

This data structure makes for an effective example, but data structures found in practice are much more complicated

Model

General case: Observe n i.i.d. copies of random variable O with probability distribution P_0

The data-generating distribution P_0 is also known to be an element of a statistical model M : $P_0 \in M$

A **statistical** model M is the set of possible probability distributions for P_0; it is a collection of probability distributions

If all we know is that we have n i.i.d. copies of O, this can be our statistical model, which we call a non-parametric statistical model

Target Parameters

Define the parameter of the probability distribution P as function of P : $\Psi(P)$

In a causal inference (specificly the Neyman-Rubin) framework, a target parameter for the effect of A would be

$$\Psi(P_0) = E_{W,0}\left[E_0(Y|A=1,W) - E_0(Y|A=0,W)\right]$$

Note that this requires additional, nontestable assumptions to infer causality

- $(A \perp Y_a|W)$; SUTVA; Positivity
- Does not change the statistical model M

Estimators

The target parameter $\Psi(P_0)$ depends on P_0 through the conditional mean $\overline{Q}_0(A,W) = E_0(Y|A,W)$ and the marginal distribution $Q_{W,0}$ of W; or

$$\overline{Q}(A,W) = E(Y|A,W) / \overline{Q}(W) = E(Y|W)$$

Where $\overline{Q}$ is an **estimator** of $\overline{Q}_0(A,W)$, shortened to $\overline{Q}_0$

An **estimator** is an algorithm that can be applied to any empirical distribution to provide a mapping from the empirical distribution to the parameter space

- But which algorithm?

Effect Estimation vs. Prediction

Both **effect** and **prediction** research questions are inherently *estimation* questions, but they are distinct in their goals

- **Prediction**: Interested in generating a function to input covariates and predict a value for the outcome: $E_0\left(Y|W\right)$
- **Effect**: Interested in estimating the true effect of exposure on outcome adjusted for covariates, $\Psi\left(P_0\right)$, the **targeted estimand**
- Targeted maximum likelihood estimation (TMLE), is an iterative procedure that updates an initial (super learner) estimate of the relevant part $\overline{Q}_0$ of the data generating distribution P_0
- See second presentation given on April 12 to the Ann Arbor ASA Chapter

Prediction: Key Concepts

1. **Loss-based estimation**: Use **loss functions** to define best estimator of $E_0\left(Y|W\right)$ & evaluate it
2. **Flexible Estimation**: Allow **data** to drive your **estimates**, but in an honest (cross validated) way
3. **Cross Validation**: Available data is partitioned to **train** and **validate** our estimators

These are detailed topics; we'll cover core concepts

Loss-Based Estimation

In order to choose a "best" algorithm to estimate $\overline{Q}_0$, must have a way to define what "best" means

Given the data structure is $O = (W, Y) \sim P_0$, with empirical distribution P_n which places probability $1/n$ on each observed O_i, $i = 1, \ldots, n$

A loss function, $L\left(O, \overline{Q}\right)$, assigns a measure of performance to a candidate function $\overline{Q}$ when applied to an observation O

One of the most commonly used loss functions is the $L2$ squared error (or quadratic) loss function:

$$L\left(O, \overline{Q}\right) = \left(Y - \overline{Q}\left(W\right)\right)^2$$

The Parameter of Interest

We define our parameter of interest, $\overline{Q}_0 = E_0(Y|W)$, as the minimizer of the expected squared error loss:

$$\overline{Q}_0 = \arg\min_{\overline{Q}} E_0 L\left(O, \overline{Q}\right)$$

$E_0 L\left(O, \overline{Q}\right)$, which we want to be small, evaluates the candidate $\overline{Q}$, and it is minimized at the optimal choice of $\overline{Q}_0$

We refer to expected loss as the **risk**

We want estimator that minimizes the expectation of the squared error loss function; we want an estimator that has small bias and variance

Flexible Estimation

The Super Learner algorithm finds the combination of algorithms minimizing the cross-validated risk

For a given problem, a "library" of candidate prediction algorithms can be proposed

- *Recommend* using a diverse set of learners (Linear Model, Support Vector Machine, Random Forest, Neural Net, etc.)
- Multi-step algorithm involving screening covariates and optimizing tuning parameters
- As long as the algorithm takes the observed W and outputs a predicted Y

No need to decide beforehand which algorithm to use; can use several by incorporating **cross-validation**

Cross-Validation

Consider a candidate library of C prediction algorithms $\{\overline{Q}_1, \ldots, \overline{Q}_C\}$

In V-fold cross-validation, the observed data O_i, $i = 1, \ldots, n$, is referred to as the learning set and is partitioned into V sets of size $\approx n/V$

For any given fold, $V - 1$ sets comprise a training set and remaining 1 set is a validation set

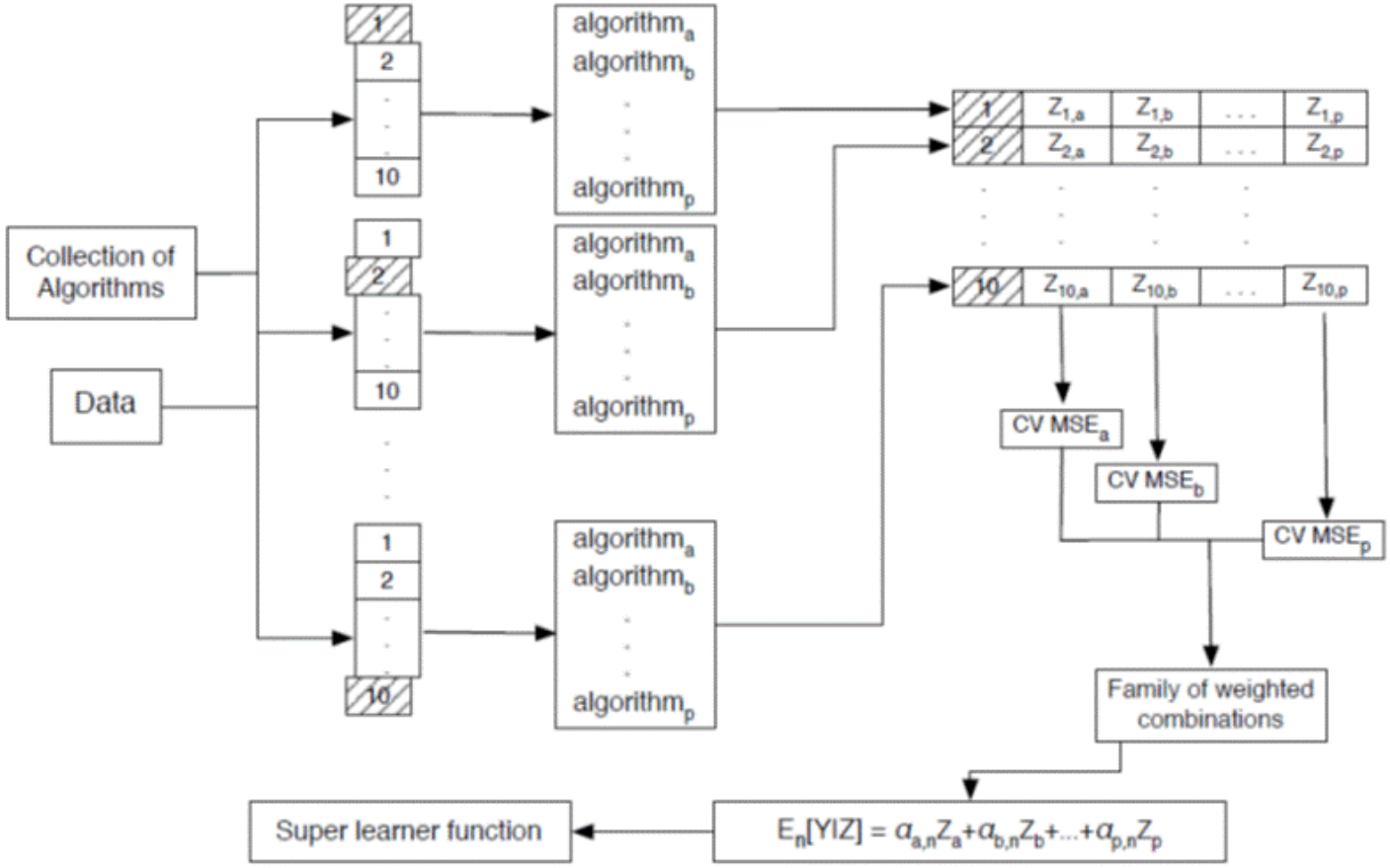
For each candidate algorithm in C, V-fold cross-validation is used to generate n cross-validated predicted values associated with the C^{th} learner, the results are stored in a $n \times C$ matrix, Z

Super Learner: Ensembling

Super Learner is a "generalized stacking" ensemble learning technique

- A second-level learner, or a *metalearner*
- It is trained with Z, the cross-validated predicted values from each candidate algorithm
- Used as inputs in a working statistical model to predict the original Y
- Use the least squares algorithm to solve for $(\alpha_1, \ldots, \alpha_C)$, the weight vector that minimizes

$$\sum_{i=1}^{n} \left(Y_i - \sum_{c=1}^{C} \alpha_c z_{ci} \right)^2$$


Collection of Algorithms
Data
1
2
10
algorithm$_a$
algorithm$_b$
algorithm$_p$
$Z_{1,a}$ $Z_{1,b}$ ... $Z_{1,p}$
$Z_{2,a}$ $Z_{2,b}$ ... $Z_{2,p}$
$Z_{10,a}$ $Z_{10,b}$ ... $Z_{10,p}$
CV MSE$_a$
CV MSE$_b$
CV MSE$_p$
Family of weighted combinations
$E_n[Y|Z] = \alpha_{a,n}Z_a+\alpha_{b,n}Z_b+...+\alpha_{p,n}Z_p$
Super learner function

Super Learner: Ensembling

Super Learner is an *oracle selector*, defined as the estimator, among all possible weighted combinations of the C prediction functions, that minimizes risk under the true data-generating distribution

- 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

Application of super learner

The SuperLearner R Package

Created by Eric Polley, National Cancer Institute

Table: Main functions in the SuperLearner package

Function	Description
`SuperLearner`	fits super learner
`CV.SuperLearner`	cross-validate in super learner
`listWrappers`	returns list of wrappers in package
`write.SL.template`	prediction wrapper template
`write.screen.template`	screening wrapper template
`write.method.template`	method wrapper template

CV SuperLearner

```
pm1 <- CV.SuperLearner(Y=Y,
                       x=x,
                       V=10,
                       family=binomial(),
                       SL.library=SL.library,
                       method="method.NNLS",
                       verbose = TRUE,
                       control = list(saveFitLibrary = TRUE),
                       cvControl = list(V=10), saveAll = TRUE,
                       parallel = 'multicore')
```

SuperLearner Arguments

Required:

- `Y` - The outcome
- `X` - The covariates
- `V` - The number of folds for the metalearner
- `family` - 'gaussian' or 'binomial' to describe the error distribution
- `SL.library` - a character vector or a list of prediction algorithms
- `method` - Method used by metalearner to combine the individual algorithms. The default is non-negative least squares

SuperLearner Arguments

Optional:

- `verbose` - helpful to set this to `TRUE` to see the progress of the estimation
- `control` - Parameters to control the estimation process, like saving the fit for each algorithm
- `cvControl` - controls the cross-validation process of the individual prediction algorithms, i.e. the number of splits for the V-fold cross-validation
- `parallel` - Options for parallel computation of the V-fold step
  Can be platform specfic!

Canidate algorithms in SuperLearner

There are two types of candidate algorithms that can be used in `SL.library`:

1. **Prediction algorithms**: Algorithms that take as input `X` and `Y` and return a predicted Y value
2. **Screening algorithms**: Algorithms designed to reduce the dimension of X. They take as input `X` and `Y` and return a logical vector indicating the columns in `X` passing the screening

Screening algorithms can be coupled with prediction algorithms to form new prediction algorithms

Canidate algorithms in SuperLearner

There are two ways to specify the algorithms in `SL.library`:

1. A character vector:
   `c("SL.glm", "SL.glmnet"", "SL.gam")`
2. A list of character vectors:
   `list(c("SL.glm", "screen.corP"), "SL.gam")`

If only using prediction algorithms, easier to use the first method. If using screening algorithms, the list is required

- Prediction algorithm listed first, followed by the screening algorithms
- Multiple screening algorithms can be used

Prediction algorithm wrappers in SuperLearner

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

```
## All prediction algorithm wrappers in SuperLearner:
```

```
##  [1] "SL.bayesglm"        "SL.caret"           "SL.caret.rpart"
##  [4] "SL.cforest"         "SL.earth"           "SL.gam"
##  [7] "SL.gbm"             "SL.glm"             "SL.glm.interaction"
## [10] "SL.glmnet"          "SL.ipredbagg"       "SL.knn"
## [13] "SL.leekasso"        "SL.loess"           "SL.logreg"
## [16] "SL.mean"            "SL.nnet"            "SL.nnls"
## [19] "SL.polymars"        "SL.randomForest"    "SL.ridge"
## [22] "SL.rpart"           "SL.rpartPrune"      "SL.step"
## [25] "SL.step.forward"    "SL.step.interaction" "SL.stepAIC"
## [28] "SL.svm"             "SL.template"
```

Screening algorithm wrappers in SuperLearner

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

```
## All screening algorithm wrappers in SuperLearner:
```

```
## [1] "All"
## [1] "screen.corP"           "screen.corRank"        "screen.glmnet"
## [4] "screen.randomForest"   "screen.SIS"            "screen.template"
## [7] "screen.ttest"          "write.screen.template"
```

Examining a wrapper function

```
SL.glmnet
```

```
## function (Y, X, newX, family, obsWeights, id, alpha = 1, nfolds = 10,
##     nlambda = 100, useMin = TRUE, ...)
## {
##     .SL.require("glmnet")
##     if (!is.matrix(X)) {
##         X <- model.matrix(~-1 + ., X)
##         newX <- model.matrix(~-1 + ., newX)
##     }
##     fitCV <- glmnet::cv.glmnet(x = X, y = Y, weights = obsWeights,
##         lambda = NULL, type.measure = "deviance", nfolds = nfolds,
##         family = family$family, alpha = alpha, nlambda = nlambda)
##     pred <- predict(fitCV$glmnet.fit, newx = newX, s = ifelse(useMin,
##         fitCV$lambda.min, fitCV$lambda.1se), type = "response")
##     fit <- list(object = fitCV, useMin = useMin)
##     class(fit) <- "SL.glmnet"
##     out <- list(pred = pred, fit = fit)
##     return(out)
## }
## <environment: namespace:SuperLearner>
```

Creating your own wrappers

Many algorithms are included in the package, but you may want to create your own

A few reasons to build your own wrappers:

- Want to use an algorithm not currently included
- Want to include a range of tuning parameters, not just the default
- Force variables to be used in chunk-wise fashion

Example

Creating your own wrapper to alter a tuning parameter

`glmnet` algorithm evaluates elastic net (penalized regression) models

The `alpha` tuning parameter controls the amount of shrinkage to estimated coefficients the algorithm applies, 1=LASSO, 0=Ridge

As `SL.ridge` wrapper doesn't handle binary variables, need to modify the wrapper to utilize various penalized regression models

```
SL.glmnet.0 <- function(..., alpha = 0){
  SL.glmnet(..., alpha = alpha)
  } # ridge penalty
```

Important notes for creating prediction wrappers

- Input must following naming syntax: Y, X, …
- Name of new function must be different than one already in the package
- Must return a list with 2 elements named `pred` and `fit`
- `pred` must be a vector with the predicted Y values
- `fit` can be anything if not using `predict` method, otherwise is a list with elements needed for `predict`

SuperLearner example

Predicting 30 day mortality for patients admitted to an ICU

The ARF dataset has 2490 observations and 47 variables including:

- **Demographic characteristics**, including age, gender, weight and race
- **Patient medical history**, 12 dichotomous variables for medicial conditions: MI, COPD, stroke, cancer, etc.
- **Current condition variables**, that provide information about the patient's current health status: three diagnostic scales, vital statistics and current disease status

Can we build a good prediction model of 30 day mortality?

Preparing data for SuperLearner

Only works with numeric matrices; can be specified in-line, i.e. `Y= dataset$Y`

Data must be preprocessed:

- Can only handle missingness in the outcome `Y`, `X` must be removed/imputed
- Continuous variables must be appropriately re-scaled
- Categorical variables must be appropriately dummy coded

Preparing data for SuperLearner

```
# Impute missing X values #
library("VIM")

# Scale cont vars #
library(arm)
cont <- c("age","edu","das2d3pc","aps1","scoma1","meanbp1","wblc1","hrt1",
          "resp1","temp1","pafi1","alb1","hema1","bili1","crea1","sod1",
          "pot1","paco21","ph1","wtkilo1")
arf[,cont] <- data.frame(apply(arf[cont], 2, function(x)
  {x <- rescale(x, "full")})); rm(cont) # standardizes by centering and
                                        # dividing by 2 sd

# Create dummy vars #
arf$rhc <- ifelse(arf$swang1=="RHC",1,0)
arf$white <- ifelse(arf$race=="white",1,0)
arf$swang1 <- arf$race <- NULL
```

Prepare SuperLearner

```
# Specify new SL prediction algorithm wrappers #
SL.glmnet.0 <- function(..., alpha = 0){
  SL.glmnet(..., alpha = alpha)
  } # ridge penalty

# Specify the SL library with prediction algorithms #

SL.library <- c("SL.glm","SL.bayesglm","SL.earth","SL.gam","SL.glmnet",
                "SL.glmnet.0","SL.knn","SL.step","SL.nnet")
```

Run SuperLearner prediction model

```
system.time({
  pm1 <- CV.SuperLearner(Y=arf$death,
                         X=arf[1:45],
                         V=10, family=binomial(),
                         SL.library=SL.library,
                         method="method.NNLS",
                         verbose = TRUE,
                         control = list(saveFitLibrary = TRUE),
                         cvControl = list(V=10), saveAll = TRUE,
                         parallel = 'multicore')

})[[3]] # Obtain computation time
```

Elements of the output from SuperLearner

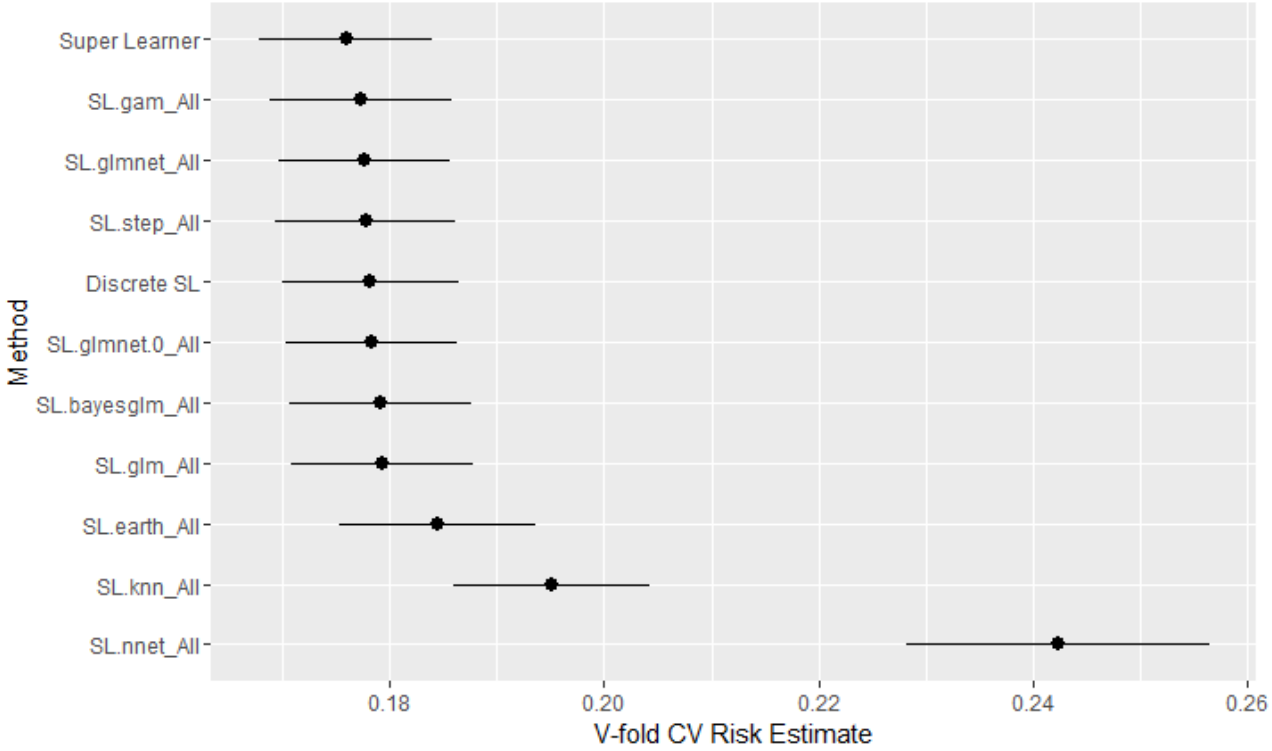
- `libraryNames` - names of algorithms in library
- `library.predict` - matrix of predicted values for Y from each algorithm
- `SL.predict` - vector of CV super learner predicted values for Y
- `whichDiscreteSL` - best algorithm for each V-fold
- `coef` - weights α for each algorithm for each V-fold
- `summary()` - Mean Squared Error (Risk) for each algorithm as well as super learner

Evaluate SuperLearner prediction model

```
summary(pm1)
```

```
      Algorithm     Ave        se     Min     Max
   Super Learner 0.17589 0.0041358 0.14915 0.19025
     Discrete SL 0.17819 0.0041751 0.14712 0.19527
      SL.glm_All 0.17927 0.0043446 0.15028 0.19577
 SL.bayesglm_All 0.17917 0.0043353 0.15014 0.19556
    SL.earth_All 0.18445 0.0046916 0.16056 0.20049
      SL.gam_All 0.17726 0.0043339 0.15272 0.19527
   SL.glmnet_All 0.17758 0.0040714 0.14712 0.19076
 SL.glmnet.0_All 0.17827 0.0040792 0.14984 0.19323
      SL.knn_All 0.19507 0.0046476 0.16739 0.20378
     SL.step_All 0.17776 0.0042912 0.14986 0.19306
     SL.nnet_All 0.24231 0.0071828 0.18667 0.31727
```

```
plot(pm1, packag ="ggplot2")
```



Best algorithm for each V-fold

```
pm1$whichDiscreteSL
```

```
## $`1`
## [1] "SL.gam_All"
##
## $`2`
## [1] "SL.glmnet_All"
##
## $`3`
## [1] "SL.glmnet_All"
##
## $`4`
## [1] "SL.step_All"
##
## $`5`
## [1] "SL.glmnet_All"
##
## $`6`
## [1] "SL.step_All"
##
## $`7`
## [1] "SL.glmnet_All"
##
```

Average of SuperLearner α weights

```
as.data.frame(colMeans(pm1$coef))
```

```
##                 colMeans(pm1$coef)
## SL.glm_All              0.000000000
## SL.bayesglm_All         0.000000000
## SL.earth_All            0.182184626
## SL.gam_All              0.238533673
## SL.glmnet_All           0.124057556
## SL.glmnet.0_All         0.000000000
## SL.knn_All              0.187070857
## SL.step_All             0.261432008
## SL.nnet_All             0.006721281
```

Plot of AUROC curves for each algorithm

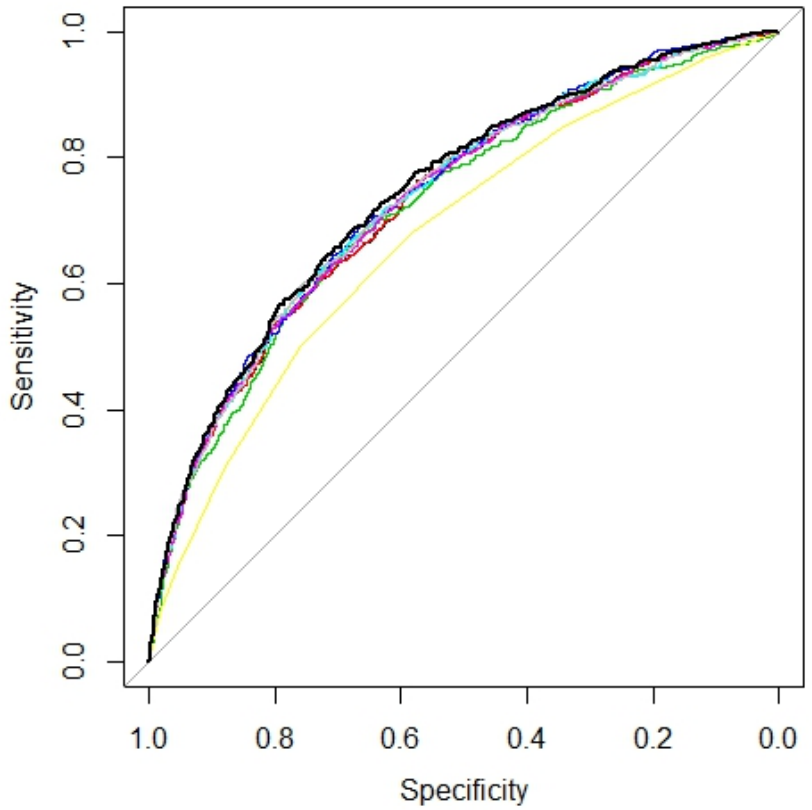


Table of AUROC curves for each algorithm

```
##                              freqta
## SL.glm_All      0.731 (0.709-0.752)
## SL.bayesglm_All 0.731 (0.709-0.752)
## SL.earth_All      0.722 (0.7-0.744)
## SL.gam_All      0.738 (0.717-0.759)
## SL.glmnet_All   0.736 (0.714-0.757)
## SL.glmnet.0_All 0.733 (0.711-0.754)
## SL.knn_All      0.677 (0.654-0.699)
## SL.step_All     0.735 (0.714-0.757)
## SL.nnet_All     0.572 (0.547-0.596)
## SL.predict      0.743 (0.722-0.764)
```

Run SuperLearner prediction model with variable screening algorithms

- Suppose you wish to limit the number of variables entered into the prediction model
- Specifically, you think that the 12 medical history variables aren't helpful
- Can generate a screening algorithm to exclude those variables:

```
screen.nohx <- function(...){
  return(c(rep(FALSE,12), rep(TRUE,33)))
  }
```

Specify the SL library with prediction and variable screening algorithms

Note that I'm only including one prediction model (generalize additive model) for demonstration purposes

```
SL.library2 <- list(c("SL.gam","All","screen.nohx","screen.glmnet"))

system.time({
pm2 <- CV.SuperLearner(Y=arf$death,
                       X=arf[1:45],
                       V=10, family=binomial(),
                       SL.library=SL.library2,
                       method="method.NNLS",
                       verbose = TRUE,
                       control = list(saveFitLibrary = TRUE),
                       cvControl = list(V=10), saveAll = TRUE,
                       parallel = 'multicore')

})[[3]]
```

Results

```
summary(pm2)
```

```
Risk is based on: Mean Squared Error

All risk estimates are based on V =  10
```

Algorithm	Ave	se	Min	Max
Super Learner	0.17643	0.0042621	0.15968	0.19481
Discrete SL	0.17730	0.0043219	0.15640	0.19587
SL.gam_All	0.17710	0.0043222	0.16020	0.19479
SL.gam_screen.nohx	0.17762	0.0042853	0.16224	0.19560
SL.gam_screen.glmnet	0.17666	0.0042918	0.15640	0.19587

```
# Examine vars in best algorithm
out = c()
for(j in 1:10){
  out = c(out,names(pm2[["AllSL"]][[j]][["fitLibrary"]][[3]][[1]][["coefficients"]]))
  }
table(out); rm(out,j)
```

```
out
   (Intercept)          amihx          chfhx           dnr1         Female
            10              1              3             10             10
         gastr       gibledhx           hema        liverhx        malighx
             7              4             10             10             10
          meta          neuro        psychhx        renalhx           resp
             3              7              7              1              3
           rhc      s(age, 2)     s(alb1, 2)     s(aps1, 2)    s(bili1, 2)
            10             10             10             10             10
   s(crea1, 2) s(das2d3pc, 2)      s(edu, 2)    s(hema1, 2)     s(hrt1, 2)
             7             10              4              2              4
 s(meanbp1, 2)   s(paco21, 2)    s(pafi1, 2)    s(resp1, 2)   s(scoma1, 2)
             7              5             10              1             10
    s(sod1, 2)    s(temp1, 2)    s(wblc1, 2)          seps        transhx
             4              5              2              6              4
        trauma          white
             3              1
```

Some (hopefully) helpful comments

Basic observations

- Super Learner is not a magic bullet: crappy data = crappy predictions
- The screening algorithms seem to work better with many V folds and lots of data, the “chunk wise” variable selection method works best with smaller data sets
- Theory says you should use many prediction algorithms, but this can be computationally demanding

Computation issues

Example computation times on different computers/R configurations

- Windows 10 laptop, 8 GB RAM, 4 logical cores/ R 3.02: 67 minutes
- UNIX desktop, 16 GB of RAM, 4 cores/ Microsoft Open R 3.02: 44 minutes
- UNIX desktop, 16 GB of RAM, 8 logical cores/ R 3.02: 31 minutes
- UNIX workstation, 64 GB of RAM, 16 logical cores/ R 3.02: 30 minutes

Super Learner R package software

- Written in R in 2010
- Candidate algorithms are trained independently
- Can parallelize the cross-validation step via multicore or SNOW
- The meta-learning step has a small computational burden
- Possible to run out of memory while working with large training sets

Two other options available

subsemble R package:

- Partitions the full dataset into subsets of observations
- Fits a specified underlying algorithm on each subset
- Uses a unique form of V-fold cross-validation to combine the subset-specific fits
- Using the UNIX workstation, computation time for the example went from 30 min to 1.6 min
- Still in development, does not provide CV risk estimates

h2oEnsemble R package: makes use of the h2o package, the R interface to the H2O platform, all training and data processing are performed in the high-performance H2O cluster

Thank you!

References

- van der Laan, M.J. and Rubin, D. (2006), Targeted Maximum Likelihood Learning. *The International Journal of Biostatistics*, 2(1). http://www.bepress.com/ijb/vol2/iss1/11/
- van der Laan, Mark J.; Polley, Eric C.; and Hubbard, Alan E., "Super Learner" (July 2007). *U.C. Berkeley Division of Biostatistics Working Paper Series*. Working Paper 222. http://biostats.bepress.com/ucbbiostat/paper222
- van der Laan, M.J. and Rose, S. *Targeted Learning: Causal Inference for Observational and Experimental Data*. Springer, Berlin Heidelberg New York, 2011. http://www.targetedlearningbook.com/
- Sapp, S.; van der Laan, M.J. and Canny, J. *Journal of Applied Statistics* (2013). Subsemble: An ensemble method for combining subset-specific algorithm fits. http://www.tandfonline.com/doi/abs/10.1080/02664763.2013.864263

Software and online resources

- *SuperLearner: Super Learner Prediction* https://cran.r-project.org/web/packages/SuperLearner/index.html
- *tmle: Targeted Maximum Likelihood Estimation* https://cran.r-project.org/web/packages/tmle/index.html
- *subsemble: An Ensemble Method for Combining Subset-Specific Algorithm Fits* https://cran.r-project.org/web/packages/subsemble/index.html
- M. Petersen and L. Balzer. *Introduction to Causal Inference*. UC Berkeley, August 2014. http://www.ucbbiostat.com/
- This presentation, the data (with documentation) and R code is available at: https://github.com/sfgrey/Super-Learner-Presentation.git