R Lab 4 - Super Learner!

Laura B. Balzer

Biostat 683 - Intro. to Causal Inference

Goals:
1. Code Discrete Super Learner (a.k.a., the cross-validation selector) to select the estimator with the lowest cross-validated risk from a library of candidate algorithms.
2. Use the `SuperLearner` package to build the best convex combination of predictions from the candidate algorithms.
3. Evaluate the performance of Super Learner with `CV.SuperLearner`.

Next lab:
We will implement targeted minimum loss-based estimation (TMLE).

1 Background - Pirate Prediction!

You have been hired by the Queen to build the best predictor of a pirate's ship success at finding buried treasure Y. Your findings are the preliminary step in a potential intervention to increase the number ships returning with treasure. The Queen has paid a handsome fee to the harbor master at Tortuga for his data. Specifically, you have data on the following predictor variables:

- W1 - the possession of a treasure map (yes:1; no:0)
- W2 - the ship's supply of rum when returning to port (in gallons)
- W3 - the number of mutinies in the last year (min:0; max: 5 or more)
- W4 - the proportion of ship members who own a parrot, have a peg leg, and/or have an eye patch

Let $W = \{W1, W2, W3, W4\}$ represent the vector of these predictors.

Image 1: http://www.jokeroo.com/pictures/dogs/971264.html

Image2: http://www.charlierandallpetfoundation.org/pet-tips/keep-your-pet-safe-on-halloween/attachment/pirate-dog-costume2/

2 Import the data set `RLab4.SuperLearner.csv`

1. **Use the `read.csv` function to read in the data set and assign it to object `ObsData`.**

   ```
   > ObsData<- read.csv("RLab4.SuperLearner.csv")
   ```

   For this to work, the `RLab4.SuperLearner.csv` file needs to be in your working directory, which can be accessed with the `getwd()` function. Alternatively, you can tell `R` the full path of the file. For example, if the `RLab4.SuperLearner.csv` file was downloaded to my desktop, I could use the following command:

   ```
   > ObsData<- read.csv("/Users/laura/Desktop/RLab4.SuperLearner.csv")
   ```

2. **Use the `names`, `head` and `summary` functions to explore the data.**

3. **Use the `nrow` function to count the number of ships in the data set. Assign it to `n`.**

3 The curse of dimensionality!

Suppose your Queen believes that the only predictors of a ship's success are possession of a treasure map ($W1$) and the number of mutinies in the last year ($W3$). Since these are categorical characteristics, she demands that you estimate the conditional probability of finding treasure, given these characteristics, $\mathbb{E}_0(Y|W1, W3) = \mathbb{P}_0(Y = 1 \mid W1, W3)$ with the non-parametric maximum likelihood estimator (NPMLE).

1. **Use the `table` function to count the number of ships within strata of map possession $W1$ and mutinies $W3$.**
   Hint: Columns in the data frame `ObsData` can be accessed with the `$` operator.

2. **Are you wary of predicting success via the sample proportion in each strata (i.e., the NPMLE)?** For example, try estimating the conditional probability of finding treasure without a map and with five counts of mutiny: $\mathbb{E}_0(Y|W1 = 0, W3 = 5) = \mathbb{P}_0(Y = 1 \mid W1 = 0, W3 = 5)$.

Take-home message: In many data applications, our statistical model $\mathcal{M}$ is non-parameteric, but the NPMLE (i.e., a fully stratified approach) is not well-defined. There can be strata with zero or only a few observations. Thereby, we need an alternative estimation approach. We could use a lower dimensional parametric model to describe the probability of finding treasure given predictor variables. However, we often do not know enough to *a priori*-specify the correct regression. If the specified parametric regression is incorrect, our point estimates and inference may be biased. Trying a bunch of regressions, looking at the results, fiddling with the specifications, and choosing the "best" (using arbitrary criteria) also leads to biased point estimates and misleading inference. Therefore, we are going to use Discrete Super Learner to choose between a library of *a priori*-specified candidate prediction algorithms, using cross-validation, and according to our selected loss function.

Reminder: This discussion also applies to estimation of the conditional mean outcome $\mathbb{E}_0(Y|A, W)$ used in a substitution estimator (a.k.a., G-computation estimator) and for the propensity score $\mathbb{P}_0(A = 1|W)$ used in IPTW.

4 Code Discrete Super Learner to select the estimator with the smallest cross-validated risk estimate

The first step of the Super Learner algorithm (and more generally loss-based learning) is to define the target parameter $\bar{Q}_0(W) = \mathbb{E}_0(Y|W)$ as the minimizer of the expectation of a loss function:

$$\bar{Q}_0(W) = argmin_{\bar{Q}}\ \mathbb{E}_0\big[L(O, \bar{Q})\big]$$

Since the outcome is binary, we could use the L2 loss function or the negative log likelihood loss:

$$L(O, \bar{Q}) = (Y - \bar{Q}(W))^2$$
$$L(O, \bar{Q}) = -log\big[\bar{Q}(W)^Y (1 - \bar{Q}(W))^{1-Y}\big]$$

For the purposes of this lab, we are going to use the L2 loss function as our measure of performance. The expectation of the loss function under the true data generating distribution $\mathbb{P}_0$ is called the *risk*.

The second step is to define a library of candidate estimators. Suppose that prior to the analysis, pirate experts came up with the following candidate estimators to include in the library for the Discrete Super Learner (i.e., the cross-validation selector):

$$\bar{Q}^a(W) = logit^{-1}\big[\beta_0 + \beta_1 W1 + \beta_2 W3 + \beta_3 W1^* W3 + \beta_5 W4^2\big]$$
$$\bar{Q}^b(W) = logit^{-1}\big[\beta_0 + \beta_1 W1 + \beta_2 log(W2) + \beta_3 W3 + \beta_4 W4 + \beta_5 W3^* W4\big]$$
$$\bar{Q}^c(W) = logit^{-1}\big[\beta_0 + \beta_1 W1 + \beta_2 W2 + \beta_3 W4 + \beta_4 W1^* W2 + \beta_5 W1^* W4 + \beta_6 W2^* W4 + \beta_7 W1^* W2^* W4\big]$$
$$\bar{Q}^d(W) = logit^{-1}\big[\beta_0 + \beta_1 W1 + \beta_2 sin(W2^2) + \beta_3 W1^* sin(W2^2) + \beta_4 log(W4)\big]$$

where $W = (W1, W2, W3, W4)$. Therefore, our library consists of four parametric regression, denoted with the superscripts $a - d$. Using cross-validation, we can generate an "honest" estimate of risk for each candidate $\bar{Q}(W) = \mathbb{E}(Y|W)$. We will choose the candidate estimator with the smallest cross-validated risk estimate. Here, we are going to select the estimator with the lowest cross-validated mean squared prediction error.

Caveat: In most real-data applications, we would not have the knowledge to *a priori*-specify transformations using trig functions. This is for illustration of how we could encode subject matter knowledge into our parametric regressions. (For a real example, see https://academic.oup.com/cid/article/71/9/2326/5614347 where we included pre-specified interactive terms to improve our Super Learner.) The corresponding homework will expand our library to include both expert-specified, parametric approaches as well as non-parametric machine learning algorithms.

1. **Which algorithm do you think will do best at predicting the pirate ship's success at finding treasure?**

2. **Create the following transformed variables and add them to data frame `ObsData`.**

```
> W2sq <- ObsData$W2*ObsData$W2
> sinW2sq<- sin(W2sq)
> logW2<- log(ObsData$W2)
> W4sq <- ObsData$W4*ObsData$W4
> sinW4sq <- sin(W4sq)
> logW4 <- log(ObsData$W4)
```

3. **Split the data into $V = 10$ folds.** With $n = 1000$ observations total, we want $n/10 = 100$ in each fold. For simplicity let us define the first hundred observations to be the first fold, the second hundred to be the second fold, and so forth.
   Hint: We can create the vector `Fold` by with the `c()` and `rep()` functions.

```
> Fold<- c(rep(1, 100), rep(2, 100), rep(3, 100),rep(4, 100),rep(5, 100),
+   rep(6, 100),rep(7, 100), rep(8, 100), rep(9, 100), rep(10, 100))
```

   Alternatively, you could use the `sample` function (without replacement) to get 10 folds of size 100.

4. **Create a matrix `Pred` with 1000 rows for the ships and 4 columns to hold the cross-validated predictions for each observation according to each candidate algorithm.**

```
> Pred <- matrix(NA, nrow=1000, ncol=4)
```

5. **Create a matrix `CV.risk` with 10 rows for the folds and 4 columns to hold the cross-validated risk estimates for each algorithm.**

6. **To implement Discrete Super Learner, use a `for` loop to fit each estimator on the training set (9/10 of the data); predict the probability of finding treasure for the corresponding validation set (1/10 of the data), and evaluate the cross-validated risk.**

   (a) **Since each fold needs to serve as the training set, have the `for` loop run from `V` is 1 to 10.** First, the observations in `Fold=1` will serve as the validation set and other 900 observations as the training set. Then the observations in `Fold=2` will be the validation set and the other 900 observations as the training set... Finally, the observations in `Fold=10` will be the validation set and the other 900 observations as the training set.

   (b) **Create the validation set as a data frame `valid` consisting of observations with `Fold` equal to `V`.**
   Hint: Use the logical `==` to select the rows of `ObsData` with `Fold==V`.

   (c) **Create the training set as a data frame `train` consisting observations with `Fold` not equal to `V`.**
   Hint: Use the logical `!=` to select the rows of `ObsData` with `Fold!=V`.

   (d) **Use `glm` to fit each algorithm on the training set.** Be sure to specify `family= 'binomial'` for the logistic regression and `data` as the training set `train`.

   (e) **For each algorithm, predict the probability of finding treasure for each ship in the validation set.** Be sure to specify the `type='response'` and `newdata` as the validation set `valid`.

   (f) **Save the cross-validated predictions for each ship in the validation set at the appropriate row in the matrix `Pred`.**

   (g) **Estimate the cross-validated risk estimate for each algorithm based on the L2 loss function.** Take the average of the squared difference between the outcomes Y in the validation set and their cross-validated predicted probabilities. **Assign the cross-validated risks as a row in the data frame `CV.risk`.**

7. **Select the algorithm with the lowest average cross-validated risk across the folds.** Apply the `colMeans` function to the matrix `CV.risk`.

8. **Fit the "chosen" algorithm on all the data.**

9. **How can we come up with an even better prediction function than the one selected?**

10. See the Bonus for hand-coding of the ensemble Super Learner.

5 Use the `SuperLearner` package to build the best combination of algorithms.

1. **If you have not done so already, install the `SuperLearner` package.** In RStudio, click on `Tools` in the menu and select `Install Packages...` Search for `SuperLearner`. Check the box for `Install Dependencies` and click `Install`.

2. **Load the Super Learner package with the `library` function.**
   ```
   > library('SuperLearner')
   ```

3. **Read the help file on `SuperLearner`.**
   ```
   > ?SuperLearner
   ```

4. **The `SuperLearner` package uses wrapper functions. Use the `listWrappers()` function to see built-in candidate algorithms. For example, explore the wrapper function for stepwise regression `SL.step`.**

   ```
   > SL.step
   ```

5. **Use the `source` function to load script file `RLab4.SuperLearner.Wrappers.R`, which includes code for the wrapper functions for the *a priori*-specified parametric estimators.**

   ```
   > source('RLab4.SuperLearner.Wrappers.R')
   ```

6. **Specify the algorithms to be included in SuperLearner's library.** Create the following vector `SL.library` of *character strings*:

   ```
   > SL.library<- c('SL.glm.EstA', 'SL.glm.EstB', 'SL.glm.EstC', 'SL.glm.EstD')
   ```

   Note: We are choosing these simple algorithms in the interest of time. Remember our Spice Girls principle; there are a lot more fun and exciting algorithms out there. Feel free to try out others.

7. **Create data frame `X` with the predictor variables.** Include the original predictor variables as well as the transformed variables.
   Hint: The following code uses the `subset` function to select all columns, but `Y`:

   ```
   > X<- subset(ObsData, select= -Y )
   ```

8. **Set the seed to 1.**

9. **Run the `SuperLearner` function. Be sure to specify the outcome `Y`, the predictors `X`, the library `SL.library` and the `family`. Also include `cvControl=list(V=10)` in order to get 10-fold cross-validation.**

   ```
   > SL.out<- SuperLearner(Y=ObsData$Y, X=X, SL.library=SL.library, family='binomial',
   +     cvControl=list(V=10))
   ```

10. **Which algorithm had the lowest cross-validated risk? Which algorithm was given the largest weight when building the convex combination of algorithms? Are the cross-validated risks from `SuperLearner` close to those obtained by your code?**

6 Evaluate the performance of Super Learner.

For a "honest" measure of performance, we add another layer of cross-validation to Super Learner.

1. **Read the help file on `CV.SuperLearner`.**

   ```
   > ?CV.SuperLearner
   ```

2. **Evaluate the performance of the Super Learner algorithm by running `CV.SuperLearner`.** Again be sure to specify the outcome `Y`, predictors `X`, `family`, and `SL.library`. (Here, we are using the default settings for the cross-validation splits.) This might take a couple minutes to run.

   ```
   > CV.SL.out<- CV.SuperLearner(Y=ObsData$Y, X=X, SL.library=SL.library, family='binomial')
   ```

   This function is partitioning the data into V^*=10 folds, running the whole Super Learner algorithm in each training set (9/10 of the data), evaluating the performance on the corresponding validation set (1/10 of the data), and rotating through the folds. Each training set will, itself, be partitioned into V=10 folds in order to run `SuperLearner`.

3. **Explore the output with the `summary` function.**

7 Bonus: Coding the weights

Try the following code to create the weights from your hand-coded Super Learner when the goal is to minimize the expected L2 loss function. (If you have a different loss function, it will be a different procedure to find the optimal weights.)

```
> # Load in the nnls library
> library(nnls)
> # Create a new data frame with the observed outcome (Y) and CV-predictions from the 4 algorithms
> X<- cbind(ObsData$Y, Pred)
> head(X)
```

```
     [,1]      [,2]      [,3]      [,4]       [,5]
[1,]    0 0.2517990 0.2652065 0.2083629 0.34108956
[2,]    0 0.2644518 0.2482541 0.2180554 0.35259727
[3,]    0 0.2232686 0.2113653 0.2097259 0.28410396
[4,]    0 0.3018996 0.3335361 0.1666539 0.28909303
[5,]    0 0.1570145 0.1164404 0.2406333 0.09424833
[6,]    1 0.1717494 0.1407970 0.2055013 0.14066346
```

```
> ## 4: estimate weights using non-linear least squares
> weights <- nnls(X[,2:5], X[,1])$x
> # then normalize to sum to 1 by dividing by the sum of the weights
> alpha<-as.matrix(weights/sum(weights))
> round(alpha,3)
```

```
      [,1]
[1,] 0.000
[2,] 0.000
[3,] 0.359
[4,] 0.641
```

```
> # compare to the package's coefficients
> SL.out
```

```
Call:
SuperLearner(Y = ObsData$Y, X = X, family = "binomial", SL.library = SL.library,
    cvControl = list(V = 10))


                     Risk      Coef
SL.glm.EstA_All 0.1825828 0.0000000
SL.glm.EstB_All 0.1786306 0.0000000
SL.glm.EstC_All 0.1655594 0.3512984
SL.glm.EstD_All 0.1561779 0.6487016
```

```
> #----------------
> ## fit all algorithms to original data & generate predictions
> PredA<- predict(glm(Y~ W1*W3 + W4sq, family='binomial',
+                     data=ObsData), type='response')
> PredB<- predict(glm(Y~ W1+ logW2 + W3*W4, family='binomial',
+                     data=ObsData), type='response')
```

```
> PredC<- predict(glm(Y~ W1*W2*W4, family='binomial',
+                    data=ObsData), type='response')
> PredD<- predict(glm(Y~ W1*sinW2sq+ logW4, family='binomial',
+                    data=ObsData), type='response')
> Pred <- cbind(PredA, PredB, PredC, PredD)
> # Take a weighted combination of predictions using nnls coeficients as weights
> Y.SL <- Pred%*%alpha
> # comparing the (non-cross-validated) weighted predictions between
> # our our hand-coded ensemble SuperLearner and the package
> mean( (ObsData$Y- Y.SL)^2)

[1] 0.1508343

> mean( (ObsData$Y- SL.out$SL.predict)^2)

[1] 0.1508343
```