R Assignment 4 - Super Learner

Laura B. Balzer

Biostat 683 - Intro. to Causal Inference

Assigned: November 11, 2021
Write-ups due: Uploaded to your personal GoogleDrive folder by November 23, 2021 by 11:59pm. Please answer all questions and include relevant R code. You are encouraged to discuss the assignment in groups, but should not copy code or interpretations verbatim. Use of RMarkdown is strongly encouraged.

1 Background Story

Given the success of our previous studies, we have been hired by lead NGOs to build a prediction function for malnutrition at the community-level. The outcome of interest Y is the average mid-upper arm circumference (MUAC) of children aged 6-59 months in each community. In this age range, a MUAC less than 110 mm indicates severe acute malnutrition. (Other indicators of severe acute malnutrition include visible severe wasting, nutritional edema, and a standardized weight for height lower than 3 standard deviations from the median.)

Figure 1: `http://www.doctorswithoutborders.co.nz/education/activities/braceletoflife/index.html`

We have data on the following community-level predictor variables:

- W1 - community's access to potable water (1-yes; 0-no)
- W2 - whether the community is located in a stable region (1-yes; 0-no)
- W3 - a measure of the community's socioeconomic status (on a scale from 0-5)
- W4 - the proportion of children visiting a health center in the last year for common childhood illnesses (e.g., diarrhea and pneumonia)
- W5 - the number of health facilities or therapeutic feeding centers in a community (min=1, max=4)

Let $W = \{W1, W2, W3, W4, W5\}$ be the set of predictors.

2 Import and explore the data set `RAssign4.SL.csv`.

1. **Use the `read.csv` function to import the data set and assign it to data frame `ObsData`.**
2. **Use the `names`, `tail` and `summary` functions to explore the data.**

3. **Use the `nrow` function to count the number of communities in the data set. Assign this number as `n`.**

Solution:

```
> ObsData<- read.csv('RAssign4.SL.csv')
> # get the column names
> names(ObsData)

[1] "W1" "W2" "W3" "W4" "W5" "Y"

> # show the obsv data on the last six communities
> tail(ObsData)

    W1 W2       W3        W4 W5         Y
495  1  1 1.569264 0.6685963  1 123.04042
496  0  0 3.624783 0.1655012  2  95.48621
497  0  1 3.972978 0.4715487  4 130.97578
498  0  1 3.408959 0.7277081  4 133.24344
499  0  0 3.329543 0.2988016  2  96.96166
500  1  1 2.500757 0.8552955  2 132.05325

> # recall: W1-water, W2-stable, W3-SES, W4-sick, W5-facilities
> # Y - ave MUAC
> summary(ObsData)

       W1              W2              W3                W4
 Min.   :0.000   Min.   :0.000   Min.   :0.01922   Min.   :0.1194
 1st Qu.:0.000   1st Qu.:0.000   1st Qu.:1.23607   1st Qu.:0.2873
 Median :0.000   Median :1.000   Median :2.43918   Median :0.3952
 Mean   :0.088   Mean   :0.518   Mean   :2.42584   Mean   :0.4127
 3rd Qu.:0.000   3rd Qu.:1.000   3rd Qu.:3.56510   3rd Qu.:0.5390
 Max.   :1.000   Max.   :1.000   Max.   :4.99842   Max.   :0.8765
       W5              Y
 Min.   :1.000   Min.   : 79.45
 1st Qu.:1.000   1st Qu.: 93.16
 Median :2.000   Median :102.21
 Mean   :1.952   Mean   :103.46
 3rd Qu.:2.000   3rd Qu.:110.72
 Max.   :4.000   Max.   :154.03

> # assign the number of communities to random variable n
> n<- nrow(ObsData)
> n

[1] 500
```

3 Code discrete Super Learner to select the estimator with the lowest 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:

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

We will use the L2 loss function:

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

but note many other loss functions are possible. The expectation of the loss function is called the *risk*. The second step is to define a library of candidate estimators. Suppose that before beginning the analysis we talked to subject matter experts and came up with the following candidate estimators for the conditional expectation of MUAC, given the predictors:

$$\begin{aligned}
\bar{Q}^a(W) &= \beta_0 \\
\bar{Q}^b(W) &= \beta_0 + \beta_1 W1 + \beta_2 W2 + \beta_3 W3 + \beta_4 W4 + \beta_5 W5 \\
\bar{Q}^c(W) &= \beta_0 + \beta_1 W1 + \beta_2 W2 + \beta_3 W3 + \beta_4 W4 + \beta_5 W5 + \beta_6 W2^* W5 \\
\bar{Q}^d(W) &= \beta_0 + \beta_1 W1 + \beta_2 W2 + \beta_3 W3 + \beta_4 W4 + \beta_5 W5 + \beta_6 W1^* W3 + \beta_7 W2^* W5
\end{aligned}$$

Therefore, our library consists of 4 parametric regressions, denoted with the superscripts $a - d$. The first algorithm corresponds to the simple mean and the second to main terms regression. The third and fourth algorithms include key interaction terms. We will choose the candidate estimator with the smallest cross-validated risk estimate. In other words, we are going to select the estimator with the lowest cross-validated mean squared prediction error.

1. **Briefly discuss the motivation for using discrete Super Learner (a.k.a., the cross-validation selector).**
2. **Set the seed to 1, and then split the data into $V = 20$ folds.** With $n = 500$ observations total, we want $n/20 = 25$ observations in each fold.
3. **Create a matrix `Pred` with 500 rows for the communities and 4 columns to hold the cross-validated predictions for each community according to each candidate algorithm.**
4. **Create an empty matrix `CV.risk` with 20 rows and 4 columns for each algorithm, evaluated at each fold.**
5. **Use a `for` loop to fit each estimator on the training set (19/20 of the data); predict the expected MUAC for the communities in the validation set (1/20 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 20.** First, the observations in $Fold = 1$ will serve as the validation set and other 475 observations as the training set. Then the observations in $Fold = 2$ will be the validation set and the other 475 observations as the training set... Finally, the observations in $Fold = 20$ will be the validation set and the other 475 observations as the training set.
   (b) **Create the validation set as a data frame `valid`, consisting of observations with `Fold` equal to `V`.**
   (c) **Create the training set as a data frame `train`, consisting of observations with `Fold` not equal to `V`.**
   (d) **Use `glm` to fit each algorithm on the training set. Be sure to specify `data=train`.**
   (e) **For each algorithm, predict the average MUAC for each community in the validation set. Be sure to specify the `type='response'` and `newdata=valid`.**

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

(g) **Estimate the cross-validated risk for each algorithm with the L2 loss function.** Take the average of the squared differences between the observed outcomes Y in the validation set and the predicted outcomes. **Assign the cross-validated risks as a row in the matrix `CV.risk`.**

6. **Select the algorithm with the lowest average cross-validated risk across the folds.** Hint: use the `colMeans` function.

7. **Fit the chosen algorithm on all the data.**

8. **How can we come up with an even better prediction function than the one selected?** (This question is not about improving computing in `R`, but about improving our ability to predict the outcome.)

Solution:

1. We do not know *a priori* which estimator will perform best. Therefore, we are setting up a competition between a set of pre-specified candidate algorithms. By specifying a loss function, we can define our measure of "best" or optimal performance. Specifically, we choose a loss function, whose expectation is minimized by the true value.

   An estimate of the risk, based on evaluating the empirical mean squared prediction error on the same data used to fit each algorithm, will underestimate the true risk. Using V-fold cross-validation, we fit each candidate algorithm on the training set and evaluate it on the validation set, which is independent data from the same distribution. This provides us with a better estimate of risk. It also helps us avoid over-fitting in the sense that the better risk estimate helps us avoid selecting an candidate estimator that is over-fitting.

```
> set.seed(1)

> ################
> # 2. Split the data into V = 20 folds
> # create the vector Fold
> Fold<- c(rep(1, 25), rep(2, 25), rep(3, 25),rep(4, 25),rep(5, 25),
+   rep(6, 25),rep(7, 25), rep(8, 25), rep(9, 25), rep(10, 25),
+   rep(11, 25), rep(12, 25), rep(13, 25),rep(14, 25),rep(15, 25),
+   rep(16, 25),rep(17, 25), rep(18, 25), rep(19, 25), rep(20, 25))
> # shuffling these up so that first 25 are not always in fold 1...
> Fold <- sample(Fold)

> # 3. create a matrix to hold the cross-validated predictions
> Pred <- matrix(NA, nrow=n, ncol=4)

> # 4. create a matrix CV.risk of size 20 by 4
> CV.risk<- matrix(NA, nrow=20, ncol=4)
> # label the columns for the candidate estimators
> colnames(CV.risk)<- c('EstA', 'EstB', 'EstC', 'EstD')

> ## Implementing discrete Super Learner
> # 5. use a for loop to fit each estimator on the training set,
> # predict the outcome on the validation set,
> # estimate the cross-validated risk....
>
```

```
> for(V in 1:20){
+
+   # b. create the validation set by finding the communities (rows) in Fold==V and
+   # grabbing all the data (columns)
+   valid<- ObsData[Fold==V, ]
+
+   # c. create the training set by finding the communities (rows) in Fold!=V and
+   # grabbing all the data (columns)
+   train<- ObsData[Fold !=V, ]
+
+   #sanity check when building
+   #nrow(valid) - should be 25; nrow(train) - should be 475
+
+   # d. fit each algorithm on the training set -
+   # be sure to specify the data as the training set
+
+   EstA<- glm(Y~ 1, data=train)
+   EstB<- glm(Y~ W1+W2+W3+W4+W5, data=train)
+   EstC<- glm(Y~ W1+W2+W3+W4+W5+W2*W5, data=train)
+   EstD<- glm(Y~ W1+W2+W3+W4+W5+W1*W3+W2*W5, data=train)
+
+   # e. for each algorithm predict the outcome for each community in the validation set
+   # specify newdata=valid and type=response
+
+   PredA<- predict(EstA, newdata=valid, type='response')
+   PredB<- predict(EstB, newdata=valid, type='response')
+   PredC<- predict(EstC, newdata=valid, type='response')
+   PredD<- predict(EstD, newdata=valid, type='response')
+
+   # f. save the cross-validated predictions
+   Pred[Fold==V, ] <- cbind(PredA, PredB, PredC, PredD)
+
+   # can see the difference between the predicted prob and the outcomes
+   #   for the validation set
+   # head(data.frame(Pred[Fold==V,], Y=valid$Y))
+
+   # g. estimate the cross-validated risk for each algorithm
+   # This uses the L2 loss
+   CV.risk[V,] <- colMeans( (valid$Y - Pred[Fold==V,])^2)
+
+ }

> # Average the CV.Risks across the folds  put these values into a vector
> colMeans(CV.risk)

      EstA       EstB       EstC       EstD
178.152371   5.105239   2.672652   2.148185
```

6. The simple mean (Estimator A) has a CV-MSE that is approximately 35-times higher than the other algorithms. Additionally, including interactions in Estimators C and D improved performance over the main terms regression (Estimator B). The algorithm with the lowest average cross-validated risk was Estimator D:

$$\bar{Q}^d(W) = \beta_0 + \beta_1 W1 + \beta_2 W2 + \beta_3 W3 + \beta_4 W4 + \beta_5 W5 + \beta_6 W1^* W3 + \beta_7 W2^* W5$$

```
> # 7. #Running the algorithm on all the data yielded the following
> EstD<- glm(Y~ W1+W2+W3+W4+W5+W1*W3+W2*W5, data=ObsData)
> summary(EstD)

Call:
glm(formula = Y ~ W1 + W2 + W3 + W4 + W5 + W1 * W3 + W2 * W5,
    data = ObsData)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-7.4694  -0.3773  -0.0436   0.3370   9.5960

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 70.98080    0.31849  222.87   <2e-16 ***
W1          28.04378    0.40153   69.84   <2e-16 ***
W2           3.95518    0.35222   11.23   <2e-16 ***
W3           3.99991    0.04673   85.60   <2e-16 ***
W4          16.23290    0.51077   31.78   <2e-16 ***
W5           4.05856    0.11960   33.94   <2e-16 ***
W1:W3       -1.66407    0.14144  -11.77   <2e-16 ***
W2:W5        4.02352    0.15870   25.35   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for gaussian family taken to be 1.936615)

    Null deviance: 88767.90  on 499  degrees of freedom
Residual deviance:   952.81  on 492  degrees of freedom
AIC: 1759.3

Number of Fisher Scoring iterations: 2

> # Given the covariates for a new community, we could use this function to predict the
> # average MUAC.
```

8. Potential improvements: The discrete Super Learner (a.k.a., cross-validation selector) can only do as well as the best algorithm in its library. We could potentially improve its performance by expanding the library with new algorithms (better suited to the problem) and with the full Super Learner to examine a weighted combination of algorithms.

4 Bonus: Completely Optional - Coding the weights

1. Write your own `R` code to estimate the optimal convex combination of weights using the L2 loss function. (In other words, do not use the `SuperLearner` package.)
2. Apply your weights to generate the ensemble based predictions.

Solution:

```
> # 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,]  81.61758 103.2965  79.62384  81.79572  81.57284
[2,]  93.16501 103.3659  96.02127  93.72689  93.42037
[3,] 139.34386 103.6224 141.32270 141.46251 138.82561
[4,]  89.68444 103.4016  90.42585  90.46072  90.34347
[5,] 117.36949 103.4002 115.41983 117.28622 117.32648
[6,] 100.83440 103.4399 102.22152 100.75059 100.73644

> ## 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,] 0.002
[2,] 0.001
[3,] 0.049
[4,] 0.948

> #----------------
> ## fit all algorithms to original data & generate predictions
> PredA<- predict(glm(Y~ 1,
+                     data=ObsData), type='response')
> PredB<- predict(glm(Y~ W1+W2+W3+W4+W5,
+                     data=ObsData), type='response')
> PredC<- predict(glm(Y~ W1+W2+W3+W4+W5+ W2*W5,
+                     data=ObsData), type='response')
> PredD<- predict(glm(Y~ W1+W2+W3+W4+W5+ W1*W3 + W2*W5,
+                     data=ObsData), type='response')
> Pred <- cbind(PredA, PredB, PredC, PredD)
> # Take a weighted combination of predictions using nnls coefficients as weights
> Y.SL <- Pred%*%alpha
> # calculating the (non-cross-validated) weighted MSE
> mean( (ObsData$Y- Y.SL)^2)

[1] 1.908134
```

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

1. **Load the Super Learner package with the `library` function and set the seed to** 252.
2. **Use the `source` function to load script file `Rassign4.Wrappers.R`, which includes the wrapper**

functions for the *a priori*-specified parametric regressions. *Note:* `SL.glm.EstA` should give identical results to `SL.mean`; both are taking the empirical mean. Likewise, `SL.glm.EstB` should give identical results to `SL.glm`; both are running main terms regression.

3. **Specify the algorithms to be included in Super Learner's library.** Create a vector `SL.library` of the following algorithms:

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

   Here, we are expanding the library in two ways: (1) by including these new algorithms, and (2) by searching for the best convex combination of algorithms.
   Bonus: **Very briefly describe the algorithms corresponding to `SL.ridge`, `SL.rpart`, and `SL.earth`.**

4. **Create data frame `X` with the predictor variables.**

5. **Run the `SuperLearner` function. Be sure to specify the outcome `Y`, the predictors `X` and the library `SL.library`. Also include `cvControl=list(V=20)` in order to get 20-fold cross-validation.**

6. **Explain the output to relevant policy makers and stake-holders. What do the columns `Risk` and `Coef` mean? Are the cross-validated risks from `SuperLearner` close to those obtained by your code?**

7. **Briefly why we don't (or shouldn't) use Machine Learning-based predictions of $\mathbb{E}_0(Y|A,W)$ in a simple substitution estimator or Machine Learning-based predictions of $\mathbb{P}_0(A|W)$ in inverse probability weighting?**

Solution:

```
> # 1 load Super Learner
> library('SuperLearner')
> # set seed
> set.seed(252)

> # 2 - load the wrapper code for the 4 algorithms
> source('RAssign4.Wrappers.R')
> # listWrappers()

> # 3  let's create a library
> SL.library<- c('SL.glm.EstA', 'SL.glm.EstB', 'SL.glm.EstC', 'SL.glm.EstD',
+     'SL.ridge','SL.rpart', 'SL.earth')

> # Bonus -  Briefly looking into the new algorithms
> # ridge: "Fit a linear model by ridge regression."
> ??lm.ridge

> # rpart: Recursive Partitioning and Regression Trees
> ??rpart

> # earth: multivariate adaptive regression splines
> ??earth

> # 4. data frame X of predictor variables
> X<- subset(ObsData, select=-Y)
```

```
> # 5. Call Super Learner
> SL.out<- SuperLearner(Y=ObsData$Y, X=X, SL.library=SL.library,
+                       cvControl=list(V=20))

> # 6. examine the SL.out object
> SL.out

Call:
SuperLearner(Y = ObsData$Y, X = X, SL.library = SL.library, cvControl = list(V = 20))



                       Risk       Coef
SL.glm.EstA_All 178.2073829 0.00000000
SL.glm.EstB_All   5.0267145 0.00000000
SL.glm.EstC_All   2.6163561 0.00000000
SL.glm.EstD_All   2.1257357 0.11135760
SL.ridge_All      5.0268928 0.00000000
SL.rpart_All     23.7154013 0.01259027
SL.earth_All      0.1940974 0.87605213
```

6. The `Risk` column gives the cross-validated risk estimate for each algorithm averaged across the 20 folds. This is the mean squared prediction error for each validation set averaged over the 20 folds. The `Coef` column gives the weight of each algorithm in the convex combination. The algorithm with the lowest average cross-validated risk was adaptive splines. It was also given 88% the weight when building the best combination of prediction algorithms. However, the weight given to Estimator D is not trivial (11%). A little bit of weight (1%) was also given to regression trees even though it had a higher CV-MSE.

```
> # compare with the CV risk calculated by the code from part II
> colMeans(CV.risk)

      EstA       EstB       EstC       EstD
178.152371   5.105239   2.672652   2.148185

> # the CV risks are very close. Hurray!

> # the names function will show the other obj in SL.out
> names(SL.out)

 [1] "call"              "libraryNames"      "SL.library"
 [4] "SL.predict"        "coef"              "library.predict"
 [7] "Z"                 "cvRisk"            "family"
[10] "fitLibrary"        "cvFitLibrary"      "varNames"
[13] "validRows"         "method"            "whichScreen"
[16] "control"           "cvControl"         "errorsInCVLibrary"
[19] "errorsInLibrary"   "metaOptimizer"     "env"
[22] "times"

> # for example, SL.out$SL.predict gives the predicted values for each obs
> SL.out$SL.predict
```

7. As detailed in Lecture 10, prediction is fundamentally a different goal than causal effect estimation (or more generally, statistical estimation of causally motivated parameters). Briefly, such estimators will have the wrong bias-variance trade-off for our statistical parameter $\Psi(\mathbb{P}_0)$. There is also no valid approach to obtain statistical inference for these estimators. We need something more! (TMLE is coming up next!)

6 Implement `CV.SuperLearner`

1. **Explain why we need `CV.SuperLearner`.**

2. **Run `CV.SuperLearner`.** Again be sure to specify the outcome `Y`, predictors `X`, and library `SL.library`. Specify the cross-validation scheme by including `cvControl=list(V=5))` and `innerCvControl=list(list(V=20))`. This might take a couple minutes.

   ```
   > CV.SL.out<- CV.SuperLearner(Y=ObsData$Y, X=X, SL.library=SL.library,
   +      cvControl=list(V=5), innerCvControl = list(list(V=20)) )
   ```

   This function is partitioning the data into V^*=5 folds, running the whole Super Learner algorithm in each training set (4/5 of the data), evaluating the performance on the corresponding validation set (1/5 of the data), and rotating through the folds. Each training set will itself be partitioned into V=20 folds in order to run `SuperLearner`.

3. **Explore the output.** For example, if the output object from `CV.SuperLearner` was `CV.SL.out`, run the following code.

   ```
   > # summary of the output of CV.SuperLearner
   > summary(CV.SL.out)
   > #
   > # returns the output for each call to Super Learner
   > CV.SL.out$AllSL
   > #
   > # condensed version of the output from CV.SL.out$AllSL with only the coefficients
   > # for each Super Learner run
   > CV.SL.out$coef
   > #
   > # returns the algorithm with lowest CV risk (discrete Super Learner) at each step.
   > CV.SL.out$whichDiscrete
   ```

   Only include the output from the *summary* function in your write-up, but *comment* on the other output.

Solution: 1. Super Learner is a data-adaptive algorithm. Thus far, we have used all the observed data to build the prediction function. `CV.SuperLearner` is used to evaluate the performance of Super Learner, to check against over-fitting, and to compare it with other algorithms. By adding another layer of cross-validation, we are training `SuperLearner` on a portion of the data and then evaluating its performance on a distinct portion of the data. This gets us an "honest" risk estimate.

```
> CV.SL.out<- CV.SuperLearner(Y=ObsData$Y, X=X, SL.library=SL.library,
+      cvControl=list(V=5), innerCvControl = list(list(V=20)) )

> summary(CV.SL.out)

Call:
CV.SuperLearner(Y = ObsData$Y, X = X, SL.library = SL.library, cvControl = list(V = 5),
    innerCvControl = list(list(V = 20)))

Risk is based on: Mean Squared Error

All risk estimates are based on V =  5
```

```
       Algorithm       Ave        se        Min       Max
   Super Learner   0.16504  0.030834   0.021605   0.26633
     Discrete SL   0.20665  0.041724   0.013118   0.39538
 SL.glm.EstA_All 177.72981 12.106850 165.795858 205.28670
 SL.glm.EstB_All   5.01318  0.559019   3.760743   6.71357
 SL.glm.EstC_All   2.64264  0.517460   1.220823   3.75921
 SL.glm.EstD_All   2.02790  0.435341   0.897383   3.44058
    SL.ridge_All   5.01450  0.557979   3.766932   6.69959
    SL.rpart_All  24.23097  2.005905  19.644971  33.30959
    SL.earth_All   0.20665  0.041724   0.013118   0.39538
```

`Super Learner`, using the optimal combination of algorithms, had the lowest average cross-validated risk: 0.165 on average. The discrete cross-validation selector (`Discrete SL`) corresponding to adaptive splines (`SL.earth`) had the second lowest cross-validated risk: 0.207 on average. These risk estimates were computed running the full Super Learning algorithm on $4/5^{th}$ of the data, and evaluating the risk on the remaining $1/5^{th}$ of the data, repeating across all folds, and averaging.

```
> names(CV.SL.out)

 [1] "call"               "AllSL"              "SL.predict"
 [4] "discreteSL.predict" "whichDiscreteSL"    "library.predict"
 [7] "coef"               "folds"              "V"
[10] "libraryNames"       "SL.library"         "method"
[13] "Y"

> # returns the output for each call to Super Learner
> CV.SL.out$AllSL

$`1`

Call:
SuperLearner(Y = cvOutcome, X = cvLearn, newX = cvValid, family = family,
    SL.library = SL.library, method = method, id = cvId, verbose = verbose,
    control = control, cvControl = valid[[2]], obsWeights = cvObsWeights,
    env = env)


                      Risk      Coef
SL.glm.EstA_All 181.7257549 0.0000000
SL.glm.EstB_All   4.6467845 0.0000000
SL.glm.EstC_All   2.3093545 0.0000000
SL.glm.EstD_All   1.7536267 0.1668235
SL.ridge_All      4.6486998 0.0000000
SL.rpart_All     26.7163801 0.0000000
SL.earth_All      0.2173759 0.8331765

$`2`

Call:
SuperLearner(Y = cvOutcome, X = cvLearn, newX = cvValid, family = family,
    SL.library = SL.library, method = method, id = cvId, verbose = verbose,
    control = control, cvControl = valid[[2]], obsWeights = cvObsWeights,
    env = env)
```

```
                       Risk       Coef
SL.glm.EstA_All 171.1584616 0.00107058
SL.glm.EstB_All   5.4007568 0.00000000
SL.glm.EstC_All   3.0145035 0.00000000
SL.glm.EstD_All   2.4617382 0.05876246
SL.ridge_All      5.4009030 0.00000000
SL.rpart_All     25.0690792 0.01013897
SL.earth_All      0.1286619 0.93002799

$`3`

Call:
SuperLearner(Y = cvOutcome, X = cvLearn, newX = cvValid, family = family,
    SL.library = SL.library, method = method, id = cvId, verbose = verbose,
    control = control, cvControl = valid[[2]], obsWeights = cvObsWeights,
    env = env)


                       Risk       Coef
SL.glm.EstA_All 178.5228300 0.00000000
SL.glm.EstB_All   5.4235001 0.00000000
SL.glm.EstC_All   2.9602476 0.00000000
SL.glm.EstD_All   2.4416711 0.10542172
SL.ridge_All      5.4241411 0.00000000
SL.rpart_All     23.5688072 0.01033717
SL.earth_All      0.2007281 0.88424112

$`4`

Call:
SuperLearner(Y = cvOutcome, X = cvLearn, newX = cvValid, family = family,
    SL.library = SL.library, method = method, id = cvId, verbose = verbose,
    control = control, cvControl = valid[[2]], obsWeights = cvObsWeights,
    env = env)


                      Risk       Coef
SL.glm.EstA_All 180.358406 0.00000000
SL.glm.EstB_All   4.862595 0.00000000
SL.glm.EstC_All   2.395263 0.00000000
SL.glm.EstD_All   2.089670 0.13172735
SL.ridge_All      4.863793 0.00000000
SL.rpart_All     23.622998 0.01111674
SL.earth_All      0.223734 0.85715591

$`5`

Call:
SuperLearner(Y = cvOutcome, X = cvLearn, newX = cvValid, family = family,
    SL.library = SL.library, method = method, id = cvId, verbose = verbose,
    control = control, cvControl = valid[[2]], obsWeights = cvObsWeights,
    env = env)
```

```
                      Risk       Coef
SL.glm.EstA_All 180.5582161 0.00000000
SL.glm.EstB_All   5.0709885 0.00000000
SL.glm.EstC_All   2.4504675 0.00000000
SL.glm.EstD_All   2.0325628 0.12283704
SL.ridge_All      5.0711748 0.00000000
SL.rpart_All     23.3744110 0.01463747
SL.earth_All      0.2285237 0.86252549
```

The cross-validated risk and coefficient estimates change slightly in each fold V*.

Solution: Cont...

```
> # condensed version of the above:  only the coefficients for each Super Learner run
> CV.SL.out$coef

  SL.glm.EstA_All SL.glm.EstB_All SL.glm.EstC_All SL.glm.EstD_All SL.ridge_All
1      0.00000000               0               0      0.16682354            0
2      0.00107058               0               0      0.05876246            0
3      0.00000000               0               0      0.10542172            0
4      0.00000000               0               0      0.13172735            0
5      0.00000000               0               0      0.12283704            0
  SL.rpart_All SL.earth_All
1   0.00000000    0.8331765
2   0.01013897    0.9300280
3   0.01033717    0.8842411
4   0.01111674    0.8571559
5   0.01463747    0.8625255
```

For all V*=5 folds, `Super Learner` gave Estimator A, Estimator B, Estimator C, ridge regression, and regression trees a weight of 0-1%. Estimator D received 6-17% of the weight, and adaptive splines 83-93% of the weight.

```
> # returns the discrete Super Learner- algorithm with lowest CV risk at each step.
> t( CV.SL.out$whichDiscrete )

     1              2              3              4              5
[1,] "SL.earth_All" "SL.earth_All" "SL.earth_All" "SL.earth_All" "SL.earth_All"
```

For all V* folds, the algorithm with the lowest cross-validated risk was adaptive splines. It is not always the scenario that the same algorithm "wins" every time.

Solution:

Appendix: The data was simulated as follows.

```
> set.seed(12.16)
> n <- 500
> W1 = rbinom(n,size=1, prob=0.1) # access to potable water
> W2 = rbinom(n, size=1, prob=plogis(0.2*W1)) # stable
> W3 = runif(n, min=0, max=5) # SES
> W4 = plogis(-2 +W1 + W2+ runif(n, 0, 2)) # sick
> W5 = 1+ rbinom(n, size=3, p=0.3) # health facilities
> # differential by whether or not access to potable water
> meanY.W <- 90 + 4*(W1+W2+W3+W4+W5+ W2*W5)  - W1*W3 +
+    W1*(20-15*W4) + (1-W1)*(-20+ 15*W4)
> Y = rnorm(n, mean=meanY.W, sd=0.1)
> ObsData<- data.frame(W1, W2, W3, W4, W5, Y)
> summary(ObsData)
> write.csv(ObsData, file='RAssign4.SL.csv', row.names=F)
> #rm(list=ls())
```