R Assignment 3

PH252D Fall 2013
Introduction to Causal Inference

Assigned: October 28, 2013
Due: November 6, 2013
Write-up: 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. You need to bring your own completed assignment to class.

1 Background Story

Given the success of our previous studies, we have been hired by lead NGOs to build a prediction model for community-level severe acute malnutrition. 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 socio-economic 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 $W = \{W1, W2, W3, W4\}$ be the set of predictors.

Solution:

```
> # the simulation data was generated as follows
> set.seed(12.16)
> n <- 5000
> 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
> W4sq<- W4^2
> cosW5<- cos(W5)
> Y = rnorm(n, mean=(99+10*W1+15*W2 +2*W3 -15*W4- 7*W4sq+2*cosW5 +5*W2*W5), sd=0.1)
> ObsData<- data.frame(W1, W2, W3, W4, W5, Y)
> summary(ObsData)
> write.csv(ObsData, file='Rassign3.Fa2013.csv', row.names=F)
> #rm(list=ls())
```

2 Import and explore the data set `Rassign3.Fa2013.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('Rassign3.Fa2013.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
4995  0  0 0.60873216 0.3926807  2  92.45187
4996  1  0 1.47052461 0.7134672  3  95.65791
4997  0  0 0.08428971 0.4476780  2  90.35491
4998  0  0 1.12818655 0.1597886  4  97.52296
4999  0  1 0.20743929 0.4439766  1 112.35356
5000  0  0 0.43452632 0.1213190  1  98.99388

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

       W1              W2               W3                  W4
 Min.   :0.000   Min.   :0.0000   Min.   :0.000327   Min.   :0.1192
 1st Qu.:0.000   1st Qu.:0.0000   1st Qu.:1.194527   1st Qu.:0.2839
 Median :0.000   Median :1.0000   Median :2.486190   Median :0.3980
 Mean   :0.108   Mean   :0.5082   Mean   :2.474991   Mean   :0.4174
 3rd Qu.:0.000   3rd Qu.:1.0000   3rd Qu.:3.731702   3rd Qu.:0.5522
 Max.   :1.000   Max.   :1.0000   Max.   :4.998937   Max.   :0.8807
       W5              Y
 Min.   :1.000   Min.   : 88.00
 1st Qu.:1.000   1st Qu.: 99.28
 Median :2.000   Median :109.39
 Mean   :1.897   Mean   :109.39
 3rd Qu.:2.000   3rd Qu.:119.30
 Max.   :4.000   Max.   :137.65

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

[1] 5000
```

3 Code discrete SuperLearner to select the estimator with the lowest cross-validated risk

The first step of the Super Learner algorithm (and more generally loss-based learning) is to define the target parameter $\bar{Q}_0(W) = E_0(Y|W)$ as the minimizer of the expectation of a loss function:

$$\bar{Q}_0(W) = argmin_{\bar{Q}}\ E_0\big[L(O, \bar{Q})\big]$$

Since the outcome is continuous and the target parameter is the conditional mean of the outcome Y given the covariates $W = (W1, W2, W3, W4, W5)$, we will use the L2 loss function:

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

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 you talked to subject matter experts and came up with the following candidate estimators for the conditional expectation of MUAC, given the covariates:

$$\bar{Q}_n^a(W) = \beta_0 + \beta_1 W1 + \beta_2 W2 + \beta_3 \sin(W3) + \beta_4 W4^2$$
$$\bar{Q}_n^b(W) = \beta_0 + \beta_1 W1 + \beta_2 W2 + \beta_3 W4 + \beta_4 \cos(W5)$$
$$\bar{Q}_n^c(W) = \beta_0 + \beta_1 W2 + \beta_2 W3 + \beta_3 W5 + \beta_4 W2^* W5 + \beta_5 W4^2 + \beta_6 \cos(W5)$$
$$\bar{Q}_n^d(W) = \beta_0 + \beta_1 W1 + \beta_2 W2 + \beta_3 W5 + \beta_4 W1^* W2 + \beta_5 W1^* W5 + \beta_6 W2^* W5 + \beta_7 W1^* W2^* W5$$

Therefore, our library consists of four parametric models, denoted with the superscripts $a - d$. Finally, 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 SuperLearner (a.k.a. the cross-validation selector).**

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

```
> sinW3<- sin(ObsData$W3)
> W4sq <- ObsData$W4*ObsData$W4
> cosW5 <- cos(ObsData$W5)
```

3. **Split the data into $V = 20$ folds.** With $n = 5000$ observations total, we want $n/20 = 250$ observations in each fold. **Create the vector `Fold` and add it to the data frame `ObsData`.**

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 4750 observations as the training set. Then the observations in $Fold = 2$ will be the validation set and the other 4750 observations as the training set... Finally, the observations in $Fold = 20$ will be the validation set and the other 4750 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) **Estimate the cross-validated risk for each algorithm with the L2 loss function.** Take the `mean` 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.**
   Hint: use the `colMeans` function.

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

8. **Can we do better?**

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 parameter value.

   An estimate of the risk, based on evaluating the empirical mean squared prediction error on the same data used when running 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 overfitting in the sense that the better risk estimate helps us avoid selecting an candidate estimator that is overfitting.)

```
> # 2. Transformed variable
> sinW3<- sin(ObsData$W3)
> W4sq <- ObsData$W4*ObsData$W4
> cosW5 <- cos(ObsData$W5)

> ObsData<- data.frame(ObsData, sinW3, W4sq, cosW5)
> ###############
> # 3. Split the data into V = 20 folds
> # create the vector Fold
> Fold<- c(rep(1, 250), rep(2, 250), rep(3, 250),rep(4, 250),rep(5, 250),
+   rep(6, 250),rep(7, 250), rep(8, 250), rep(9, 250), rep(10, 250),
+   rep(11, 250), rep(12, 250), rep(13, 250),rep(14, 250),rep(15, 250),
+   rep(16, 250),rep(17, 250), rep(18, 250), rep(19, 250), rep(20, 250))

> # add the Fold vector to the data frame
> ObsData<- data.frame(ObsData, Fold)

> # 4. create a matrix CV.risk of size 20 by 4
> CV.risk<- matrix(NA, nrow=20, ncol=4)

> ## Implementing discrete SuperLearner
> # 5. use a for loop to fit each estimator on the training set,
> # predict the outcome on the validation set,
> # evaluate 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 250; nrow(train) - should be 4750
+
+   # d. fit each algorithm on the training set -
+   # be sure to specify the data as the training set
+
+   EstA<- glm(Y~ W1+ W2 + sinW3 + W4sq, data=train)
+   EstB<- glm(Y~ W1+ W2 + W4+cosW5, data=train)
+   EstC<- glm(Y~ W2*W5+ W3+ W4sq +cosW5, data=train)
+   EstD<- glm(Y~ W1*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')
+
+   # can see the difference between the observed outcome and the predicted ooutcome
+   # for the validation set
+   #head(data.frame(PredA, PredB, PredC, PredD, valid$Y))
+
+   # f) estimate the cross-validated risk for each algorithm
+   # This uses the L2 loss
```

```
+
+   CV.risk[V,] <- c( mean((valid$Y - PredA)^2), mean((valid$Y - PredB)^2),
+     mean((valid$Y - PredC)^2), mean((valid$Y - PredD)^2))
+
+ }

> # 5. Average the CV.Risks across the folds  put these values into a vector
> colMeans(CV.risk)

[1]  8.312289 13.025325  7.762348 16.031240
```

6. The algorithm with the lowest average cross-validated risk was Estimator C:

$$Qbar_n^c(W) = \beta_0 + \beta_1 W2 + \beta_2 W3 + \beta_3 W5 + \beta_4 W2W5 + \beta_5 W4sq + \beta_6 \cos(W5)$$

```
> # 6. #Running the algorithm on all the data yielded the following
> EstC<- glm(Y~ W2*W5+ W3+ W4sq +cosW5, data=ObsData)
> EstC

Call:  glm(formula = Y ~ W2 * W5 + W3 + W4sq + cosW5, data = ObsData)

Coefficients:
(Intercept)           W2           W5           W3         W4sq        cosW5
   95.64175     12.84349      0.07781      2.02159    -10.80041      2.11408
      W2:W5
    4.87470

Degrees of Freedom: 4999 Total (i.e. Null);  4993 Residual
Null Deviance:          618800
Residual Deviance: 38700        AIC: 24440

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

1. **Load the SuperLearner package with the `library` function and set the seed to `252`.**

2. **Use the `source` function to load script file `Rassign3.Fa2012.Wrappers.R`, which includes the wrapper functions for the *a priori*-specified parametric regressions.**

3. **Specify the algorithms to be included in SuperLearner'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.rpartPrune', 'SL.polymars', 'SL.mean')
```

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.rpartPrune`, `SL.polymars` and `SL.mean`.**

4. **Create data frame `X` with the predictor variables.** Include the original predictor variables and the transformed 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?**

Solution:

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

> # 2 - load the wrapper code for the 4 algorithms
> source('Rassign3.Fa2013.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.rpartPrune', 'SL.polymars', 'SL.mean')

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

> # rpartPrune: Regression tree (Recursive PARTitioning) with pruning
> ??rpart

> # polymars: "An adaptive regression procedure using piecewise linear
> #  splines to model the response."
> ??polymars

> # mean: "compute a weighted mean"
> ?weighted.mean

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

> # 5. Call SuperLearner
> 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     8.312171 0.00000000
SL.glm.EstB_All    13.022424 0.00000000
SL.glm.EstC_All     7.763466 0.22657164
SL.glm.EstD_All    16.036930 0.03336734
SL.ridge_All        3.995689 0.47930974
SL.rpartPrune_All   4.833085 0.25031900
SL.polymars_All    25.427762 0.01043228
SL.mean_All       123.816749 0.00000000
```

The `Risk` column gives the cross-validated risk 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 ridge regression. It was also given the most weight when building the best combination of prediction algorithms. Nonetheless, the weights given to Estimator C and the regression tree (with pruning) are not trivial. Notice that Estimator D and `Polymars` contributed to building the optimal predictor, even though their average risks may have been larger than other algorithms (e.g. Estimator A) receiving zero weight.

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

[1]  8.312289 13.025325  7.762348 16.031240

> # the CV risks are the same (within roundoff error). Hurray!

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

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

5 Implement `CV.SuperLearner`

1. **Explain why we need `CV.SuperLearner`.**
2. **Run `CV.SuperLearner`.** Again be sure to specify the outcome `Y`, predictors `X`, library `SL.library`, folds `V=20` and `cvControl=list(V=20)`. This might take a couple minutes.

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

   This function is partitioning the data into V^*=20 folds, running the whole SuperLearner algorithm in each training set (19/20 of the data), evaluating the performance on the corresponding validation set (1/20 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 SuperLearner
> CV.SL.out$AllSL
> #
> # condensed version of the output from CV.SL.out$AllSL with only the coefficients
> # for each SuperLearner run
> CV.SL.out$coef
> #
> # returns the algorithm with lowest CV risk (discrete SuperLearner) at each step.
> CV.SL.out$whichDiscrete
```

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

Solution: 1. SuperLearner is a data-adaptive algorithm. Thus far, we have used the whole learning set (i.e. 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 set of data and then evaluating its performance on external data. This gets us an "honest" risk estimate.

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

> summary(CV.SL.out)

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


Risk is based on: Mean Squared Error

All risk estimates are based on V =  20

         Algorithm      Ave       se        Min      Max
     Super Learner   2.6188 0.060264   1.581827   6.7448
       Discrete SL   4.0011 0.073520   3.453914   4.6274
    SL.glm.EstA_All   8.3128 0.218943   7.250448  10.4334
    SL.glm.EstB_All  13.0167 0.241911  11.684601  14.2952
    SL.glm.EstC_All   7.7715 0.219762   5.777135   9.7560
    SL.glm.EstD_All  16.0216 0.268350  14.333026  17.8787
       SL.ridge_All   4.0011 0.073520   3.453914   4.6274
 SL.rpartPrune_All   4.8419 0.110220   4.009217   5.5624
    SL.polymars_All  43.3631 1.156409   0.009443 114.5606
        SL.mean_All 123.7994 1.468275 115.476111 134.6679
```

`Super Learner`, using the optimal combination of algorithms, had the lowest average cross-validated risk. The discrete cross-validation selector (`Discrete SL`) corresponding to ridge regression (`SL.ridge`) had the second lowest cross-validated risk. These risk estimates were computed running the full SuperLearning algorithm on $19/20^{th}$ of the data, and evaluating the risk on the remaining $1/20^{th}$ 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 SuperLearner
> CV.SL.out$AllSL
```

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

```
> # condensed version of the above output with only the coefficients for each SuperLearner 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               0       0.2214463      0.03189549    0.4585768
2                0               0       0.2189295      0.03915404    0.4814343
3                0               0       0.2301517      0.02986073    0.4728152
4                0               0       0.2248271      0.03545444    0.4737083
5                0               0       0.2242273      0.03282143    0.4725037
6                0               0       0.2327324      0.03898131    0.4833783
7                0               0       0.2306720      0.04132541    0.4925164
8                0               0       0.1936646      0.01479304    0.3678368
9                0               0       0.2215477      0.02792855    0.4646734
10               0               0       0.2205826      0.02745039    0.4499402
11               0               0       0.2284673      0.04046930    0.4886133
12               0               0       0.2219753      0.03419895    0.4783242
13               0               0       0.2099570      0.02475238    0.4641070
14               0               0       0.2265372      0.03377519    0.4778872
15               0               0       0.2276852      0.03289945    0.4852898
16               0               0       0.2221733      0.02904818    0.4818212
17               0               0       0.2276429      0.03304082    0.4782047
18               0               0       0.2164943      0.02094204    0.4386059
19               0               0       0.2258606      0.03882940    0.4895616
20               0               0       0.2235179      0.03304273    0.4784551
   SL.rpartPrune_All SL.polymars_All SL.mean_All
1          0.2488527    0.0392287411           0
2          0.2220348    0.0384473574           0
3          0.2551325    0.0120397887           0
4          0.2271389    0.0388712153           0
5          0.2704475    0.0000000000           0
6          0.2449081    0.0000000000           0
7          0.2351861    0.0003001418           0
8          0.2454318    0.1782737061           0
9          0.2448963    0.0409541200           0
10         0.2474241    0.0546027102           0
11         0.2424501    0.0000000000           0
12         0.2256087    0.0398928415           0
13         0.2363162    0.0648674148           0
14         0.2618005    0.0000000000           0
15         0.2509928    0.0031327936           0
16         0.2580246    0.0089326854           0
17         0.2611116    0.0000000000           0
```

```
18         0.2407590    0.0831987883           0
19         0.2405434    0.0052050336           0
20         0.2501937    0.0147904675           0

> summary(CV.SL.out$coef)

 SL.glm.EstA_All SL.glm.EstB_All SL.glm.EstC_All  SL.glm.EstD_All
 Min.   :0       Min.   :0       Min.   :0.1937   Min.   :0.01479
 1st Qu.:0       1st Qu.:0       1st Qu.:0.2212   1st Qu.:0.02877
 Median :0       Median :0       Median :0.2239   Median :0.03297
 Mean   :0       Mean   :0       Mean   :0.2225   Mean   :0.03203
 3rd Qu.:0       3rd Qu.:0       3rd Qu.:0.2277   3rd Qu.:0.03630
 Max.   :0       Max.   :0       Max.   :0.2327   Max.   :0.04133
  SL.ridge_All    SL.rpartPrune_All SL.polymars_All      SL.mean_All
 Min.   :0.3678   Min.   :0.2220    Min.   :0.0000000   Min.   :0
 1st Qu.:0.4645   1st Qu.:0.2395    1st Qu.:0.0002251   1st Qu.:0
 Median :0.4780   Median :0.2452    Median :0.0134151   Median :0
 Mean   :0.4689   Mean   :0.2455    Mean   :0.0311369   Mean   :0
 3rd Qu.:0.4822   3rd Qu.:0.2520    3rd Qu.:0.0401582   3rd Qu.:0
 Max.   :0.4925   Max.   :0.2704    Max.   :0.1782737   Max.   :0
```

For all $V^*=20$ folds, `Super Learner` gave Estimator A, Estimator B and the mean algorithm a weight of 0. Estimator C received about 22% of the weight, Estimator D about 3%, ridge regression a little less than 50% and the regression trees about 24%. In some folds, `polymars` received a non-zero weight and contributed up to 17.8% in one fold.

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

     1              2              3              4              5
[1,] "SL.ridge_All" "SL.ridge_All" "SL.ridge_All" "SL.ridge_All" "SL.ridge_All"
     6              7              8              9              10
[1,] "SL.ridge_All" "SL.ridge_All" "SL.ridge_All" "SL.ridge_All" "SL.ridge_All"
     11             12             13             14             15
[1,] "SL.ridge_All" "SL.ridge_All" "SL.ridge_All" "SL.ridge_All" "SL.ridge_All"
     16             17             18             19             20
[1,] "SL.ridge_All" "SL.ridge_All" "SL.ridge_All" "SL.ridge_All" "SL.ridge_All"
```

For all V^* folds, the algorithm with the lowest cross-validated risk was ridge regression. It is not always the case that the same algorithm "wins" every time.

6 Bonus! -Completely Optional

1. **Try adding more algorithms to the `SuperLearner` library.**
2. **Try writing your own wrapper function. The following code will give you a template.**

```
> SL.template

function (Y, X, newX, family, obsWeights, id, ...)
{
    if (family$family == "gaussian") {
    }
    if (family$family == "binomial") {
    }
    pred <- numeric()
    fit <- vector("list", length = 0)
    class(fit) <- c("SL.template")
    out <- list(pred = pred, fit = fit)
    return(out)
}
<environment: namespace:SuperLearner>
```