R Homework Three

Katherine Wolf
Introduction to Causal Inference (PH252D)
April 10, 2020

1 Background story.

2 Import and explore the data set `RAssign3.csv`.

2.1 Use the `read_csv` function to import the dataset and assign it to dataframe `obs_data`.

```
library(tidyverse)

obs_data <- read_csv("RAssign3.csv")
```

2.2 Use the `names`, `tail`, and `summary` functions to explore the data.

```
names(obs_data)

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

tail(obs_data)

## # A tibble: 6 x 6
##      W1    W2     W3    W4    W5     Y
##   <dbl> <dbl>  <dbl> <dbl> <dbl> <dbl>
## 1     0     0 0.609  0.393     2  92.5
## 2     1     0 1.47   0.713     3  95.7
## 3     0     0 0.0843 0.448     2  90.4
## 4     0     0 1.13   0.160     4  97.5
## 5     0     1 0.207  0.444     1 112.
## 6     0     0 0.435  0.121     1  99.0

summary(obs_data)

##        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
```

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

```
n <- nrow(obs_data)

n

## [1] 5000
```

3 Code discrete Super Learner to select the estimator with the lowest cross-validated risk estimate.

3.1 Briefly discuss the motivation for using discrete Super Learner (a.k.a. the cross-validation selector).

One motivation for the discrete Super Learner is to find a good statistical model of our outcome when the non-parametric maximum likelihood estimator is not well defined due to strata with zero or only a few observations, leading to over-fitting, but when we don't know enough *a priori* to specify the correct parametric model and want to choose the best among candidates. In particular, the discrete Super Learner allows us to avoid the potential bias (usually incorrect rejection of the null hypothesis as investigators prefer models that confirm their prior beliefs or offer significant results) that can arise from using ad hoc model specification procedures, as well as the corresponding misleading uncertainty estimates that assume an *a priori* specified model and ignore multiple looks at the data. The discrete Super Learner resolves these problems by specifying the candidate models and the way of choosing among them ahead of time and then incorporating the model selection process into the estimator. (The cross-validation aspect, specifially, allows the comparison of model performance on independent data from the same distribution.)

3.2 Create the following transformed variables and add them to the data frame `obs_data`:

- `sin_W3 <- sin(obs_data$W3)`
- `W4_sq <- obs_data$W4 * obs_data$W4`
- `cos_W5 <- cos(obs_data$W5)`

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

3.3 Split the data into $V = 20$ folds. Create the vector `fold` and add it to the data frame `obs_data`.

```
# With n = 5000 observations total, we want n/20 = 250 observations in each fold.

obs_data$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))
```

3.4 Create an empty matrix `CV_risk` with 20 rows and 4 columns for each algorithm, evaluated at each fold.

```
cv_risk <- matrix(NA, nrow=20, ncol=4)
```

3.5 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.

1. **Since each fold needs to serve as the training set, have the `for` loop run from `V` is 1 to 20.**
2. **Create the validation set as a data frame `valid`, consisting of observations with `fold` equal to `V`.**
3. **Create the training set as a data frame `train`, consisting of observations with `fold` not equal to `V`.**
4. **Use `glm` to fit each algorithm on the training set. Be sure to specify `data = train`.**
5. **For each algorithm, predict the average MUAC for each community in the validation set. Be sure to specify the `type = 'response'` and `newdata = valid`.**
6. **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`.**

```
for (V in 1:20){
  valid <- obs_data[obs_data$fold == V,]
  train <- obs_data[obs_data$fold != V,]
  model_a <- glm(Y ~ W1 + W2 + sinW3 + W4sq, data = train)
  model_b <- glm(Y ~ W1 + W2 + W4 + cosW5, data = train)
  model_c <- glm(Y ~ W2 + W3 + W5 + W2:W5 + W4sq + cosW5, data = train)
  model_d <- glm(Y ~ W1*W2*W5, data = train)

  predict_a <- predict(object = model_a, type = 'response', newdata = valid)
  predict_b <- predict(object = model_b, type = 'response', newdata = valid)
  predict_c <- predict(object = model_c, type = 'response', newdata = valid)
  predict_d <- predict(object = model_d, type = 'response', newdata = valid)

  cv_risk[V,1] <- mean((valid$Y - predict_a)^2)
  cv_risk[V,2] <- mean((valid$Y - predict_b)^2)
```

```r
  cv_risk[V,3] <- mean((valid$Y - predict_c)^2)
  cv_risk[V,4] <- mean((valid$Y - predict_d)^2)
}

cv_risk
```

```
##            [,1]     [,2]     [,3]     [,4]
##  [1,]  8.939385 13.81455 5.097290 16.11975
##  [2,]  7.740905 13.27176 8.472968 17.08363
##  [3,]  9.166189 13.00529 7.489641 16.24683
##  [4,]  9.583816 14.85455 7.937800 14.84962
##  [5,]  7.672768 12.55622 7.384802 14.55799
##  [6,]  7.186142 11.93893 6.704494 13.73259
##  [7,]  8.251682 13.33489 6.772499 17.17728
##  [8,]  8.862116 12.15105 8.618398 16.63734
##  [9,]  9.321865 11.83951 6.761189 15.47617
## [10,]  8.268905 14.42099 7.310155 15.28207
## [11,]  7.780033 14.25615 8.912688 16.19174
## [12,]  7.829750 11.78438 8.245043 16.37069
## [13,]  7.053946 11.18856 9.516343 16.37126
## [14,]  7.954040 13.07750 7.092210 14.95172
## [15,]  7.888961 13.18341 8.862611 18.47103
## [16,]  8.803498 12.80874 7.704269 15.58583
## [17,]  8.321003 12.62118 9.472928 16.64250
## [18,]  8.623221 13.51818 8.011789 16.69266
## [19,]  6.823230 11.80595 7.939244 16.93725
## [20,] 10.174328 15.07470 6.940609 15.24685
```

3.6 Select the algorithm with the lowest average cross-validated risk. Hint: Use the `colMeans` function.

```r
# get the average risks
average_risks <- colMeans(cv_risk)

# restore their model names
names(average_risks) <- c("model_a", "model_b", "model_c", "model_d")

# print them
average_risks
```

```
##   model_a   model_b   model_c   model_d
##  8.312289 13.025325  7.762348 16.031240
```

```r
# find the average risk that minimizes the L2 loss function
minimum_average_risk <- average_risks[average_risks == min(average_risks)]

# print it
minimum_average_risk
```

```
##  model_c
## 7.762348
```

```r
# get the model name
best_discrete_model_name <- names(minimum_average_risk)
```

3.7 Fit the chosen algorithm on all the data.

```
# fit the best model
# (Note: This code takes the name of whatever model came out on top
#  as a character string stored in the variable best_discrete_model_name,
#  deparse-substitutes it back to a variable to get the
#  glm() model object, pulls its formula using formula(), and then
#  uses that to run the new glm(), replacing
#  the data with the full dataset obs_data. I did that so that
#  I wouldn't have to check manually which model won in the prior
#  step.)
discrete_best_estimate <-
  glm(formula = formula(deparse(substitute(best_discrete_model_name))),
      data = obs_data)

discrete_best_estimate

##
## Call:  glm(formula = formula(deparse(substitute(best_discrete_model_name))),
##     data = obs_data)
##
## Coefficients:
## (Intercept)           W2           W3           W5         W4sq        cosW5
##    95.64175     12.84349      2.02159      0.07781    -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

summary(discrete_best_estimate)

##
## Call:
## glm(formula = formula(deparse(substitute(best_discrete_model_name))),
##     data = obs_data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.6682  -1.9842  -0.4801   0.9096   9.5123
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  95.64175    0.26641 359.001   <2e-16 ***
## W2           12.84349    0.21116  60.824   <2e-16 ***
## W3            2.02159    0.02716  74.424   <2e-16 ***
## W5            0.07781    0.14745   0.528    0.598
## W4sq        -10.80041    0.29801 -36.242   <2e-16 ***
## cosW5         2.11408    0.18974  11.142   <2e-16 ***
## W2:W5         4.87470    0.09881  49.335   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## (Dispersion parameter for gaussian family taken to be 7.75025)
##
##     Null deviance: 618826  on 4999  degrees of freedom
## Residual deviance:  38697  on 4993  degrees of freedom
## AIC: 24437
##
## Number of Fisher Scoring iterations: 2
```

3.8 Can we do better?

Model C had the lowest mean squared prediction error of our four discrete models, with the stability of the region ($W2$), community socioeconomic status ($W3$), the community number of health facilities ($W5$) and its cosine ($cosW5$), the square of the community proportion of children visiting a health center in the last year for a common childhood illness ($W4$), and the multiplicative interaction between region stability and the community number of health facilities ($W2 : W5$) all emerging as predictors of mid-upper-arm circumference in the model. I bet that we can do better, however, since although we specified four models that were informed by subject matter knowledge, we stipulated that we had to choose one and only one of them. A combination of models could work even better (i.e., have a lower mean squared prediction error).

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

4.1 Load the Super Learner package with the `library` function and set the seed to 252.

```
library(SuperLearner)

## Warning: package 'SuperLearner' was built under R version 3.6.3
## Loading required package: nnls
## Super Learner
## Version: 2.0-26
## Package created on 2019-10-27

set.seed(252)
```

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

```
source("Rassign3.Wrappers.R")
```

4.3 Specify the algorithms to be included in Super Learner's 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: Very briefly describe the algorithms corresponding to `SL.ridge`, `SL.rpartPrune`, `SL.polymars` and `SL.mean`.

```
SL.ridge

SL.rpartPrune

SL.polymars

SL.mean
```

- The `SL.ridge` algorithm uses the `MASS` package to fit a linear ridge regression model to the input covariates X and outcome Y (with ridge parameter λ values ranging from 1 to 20 scaled by 0.1), chooses the model with the lowest generalized cross validation (GCV) value, and outputs the chosen model's parameter estimates and its predicted X values.
- The `SL.rpartPrune` algorithm uses the `rpart` package to build a regression tree for a continuous Y or a classification tree for a binary Y, prunes the tree using the pruning parameter value CP that reults in the lowest prediction error, and outputs the chosen model's parameter estimates, including the pruning parameter value, and its predicted X values.
- The `SL.polymars` algorithm for a continuous outcome uses the `polspline` package to fit a multivariate adaptive polynomial spline regression model with knots a minimum of three order statistics apart that minimizes the residual sum of squares divided by the square of $1-(4*model\ size)/cases$ and outputs both the model parameters and the predicted X values the chosen model generates. For a binary outcome `polspline` fits a a polychotomous regression and multiple classification selected using five-fold cross validation and outputs both the chosen model's parameter estimates and its predicted X values.
- The `SL.mean` algorithm fits a very simple model by calculating the mean of the outcome Y across the observations, weighted by any supplied observation weights, and outputs its parameter estimate, which is simply that mean, and its predicted values, which are simply also that mean repeated times the number of observations.

4.4 Create data frame `X` with the predictor variables.

Include the original predictor variables and the transformed variables.

```
X <- obs_data %>%
  select(-c(Y, fold)) %>%
  as.data.frame()
```

4.5 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.

```
sl_out <- SuperLearner(Y = obs_data$Y,
                       X = X,
                       SL.library = sl_library,
                       cvControl = list(V = 20))

## Loading required namespace: polspline
## Loading required namespace: MASS
## Loading required namespace: rpart

sl_out

##
## Call:
## SuperLearner(Y = obs_data$Y, X = X, SL.library = sl_library, cvControl = list(V = 20))
##
##
```

```
##
##                       Risk       Coef
## SL.glm.EstA_All     8.315724 0.00000000
## SL.glm.EstB_All    13.030465 0.00000000
## SL.glm.EstC_All     7.765873 0.22633112
## SL.glm.EstD_All    16.028482 0.03433797
## SL.ridge_All        3.994286 0.48544923
## SL.rpartPrune_All   4.821257 0.25388168
## SL.polymars_All    26.587062 0.00000000
## SL.mean_All       123.810204 0.00000000
```

4.6 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?

`Risk`
Colloquially, the **`Risk`** column effectively gives the "score" of the model, where lower is better. Specifically, it gives the cross-validated risk for each algorithm averaged across twenty folds. More specifically, the word "risk" in the prior sentence denotes the expectation of a loss function, $\mathbb{E}_0 L(O, \bar{Q})$, where

- $O = (Y, W)$ is a random variable representing the set of observed random variables;
- Y is a random variable representing the outcome;
- W is a random variable representing the covariates $W1$, $W2$, $W3$, $W4$, and $W5$ and their transformations as calculated above;
- The subscript 0 denotes parameters of the distribution of the observed data, $(W, Y) \sim P_0$;
- $\bar{Q}$ is a candidate function for estimating the expected value of Y conditional on W, $\mathbb{E}_0(Y|W)$; and
- L denotes a loss function, or a measure of performance assigned to $\bar{Q}$.

Even more specifically, here the word "risk" denotes the expectation of the L2 loss function $L(O, \bar{Q}) = (Y - \bar{Q}(A, W))^2$, also known as the mean squared prediction error: $L(O, \bar{Q}) = \mathbb{E}_0[(Y - \bar{Q}(W))^2]$.

We define our target parameter, $\bar{Q}_0$, then, as the candidate function of the that minimizes the L2 loss function:

$$\bar{Q}_0(W) = argmin_{\bar{Q}} \mathbb{E}_0[(Y - \bar{Q}(W))^2].$$

`Coef`
The **`Coef`** column gives the weight of each algorithm in the final convex combination of algorithms that had the lowest cross-validated mean square error (risk) after regressing the outcome Y on the cross-validated predicted values of each algorithm.

Comparing the discrete and ensemble SuperLearner

```
sl_risks <- sl_out$cvRisk[1:4]

names(sl_risks) <- c("model_a", "model_b", "model_c", "model_d")

# table of both manual (average_risks) and SuperLearner (sl_risks) risk values
merged_risks <- bind_rows(average_risks, sl_risks) %>%
  add_column("algorithm" = c("manual", "SuperLearner"),
             .before = 1)

merged_risks
```

```
## # A tibble: 2 x 5
##   algorithm    model_a model_b model_c model_d
##   <chr>          <dbl>   <dbl>   <dbl>   <dbl>
## 1 manual          8.31    13.0    7.76    16.0
## 2 SuperLearner    8.32    13.0    7.77    16.0
```

From the table above, the discrete cross-validated risks from `SuperLearner` look basically identical to those obtained from my code.

5 Implement `CV.SuperLearner`.

5.1 Explain why we need `CV.SuperLearner`.

We need `CV.SuperLearner` to evaluate the performance of `SuperLearner`. It adds another layer of cross validation to (1) check for overfitting by the SuperLearning algorithm and (2) evaluate the entire `SuperLearner` algorithm against other modeling algorithms.

5.2 Run `CV.SuperLearner`.

```
cv_sl_out <- CV.SuperLearner(Y = obs_data$Y,
                             X = X,
                             SL.library = sl_library,
                             cvControl=list(V=5),
                             innerCvControl=list(list(V=20)))

## Warning in CV.SuperLearner(Y = obs_data$Y, X = X, SL.library = sl_library, : Only a single innerCvControl
is given, will be replicated across all cross-validation split calls to SuperLearner
```

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

```
summary(cv_sl_out)

##
## Call:
## CV.SuperLearner(Y = obs_data$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   2.4076 0.051714   2.058445   2.6728
##        Discrete SL   3.9984 0.073388   3.640737   4.1874
##    SL.glm.EstA_All   8.3112 0.218672   7.249236   8.9454
##    SL.glm.EstB_All  13.0328 0.242088  12.607410  13.6007
##    SL.glm.EstC_All   7.7705 0.219642   7.532044   8.1683
##    SL.glm.EstD_All  16.0284 0.268789  15.525887  16.5901
##       SL.ridge_All   3.9984 0.073388   3.640737   4.1874
##   SL.rpartPrune_All   5.0513 0.121594   4.511556   5.6756
##    SL.polymars_All  22.0382 0.826980   0.010278  94.4624
##        SL.mean_All 123.9447 1.470855 118.382591 127.9419
```

From the `summary` output, the ensemble Super Learner algorithm beat all the individual discrete parameterizations in terms of the cross-validated risk (defined here as the expectation of the L2 loss function, or the mean squared prediction error). The `whichDiscrete` output also shows that ridge regression had the lowest risk in each of the five cross-validation folds. The first-place performance (risk) of the ensemble combination of models was followed by the ridge regression algorithm and the discrete SuperLearner algorithm, exactly tied for second place because ridge regression is the best performing discrete algorithm included in the SuperLearner library. The `AllSL` and `coef` output shows that the ensemble SuperLearner algorithm consistently assigns over half the weight in each fold to the ridge regression algorithm in each of the five folds of the top layer of cross-validation, a little under a quarter of the weight to our prespecified estimator C, a little above a fifth of the weight to the regression tree algorithm, and then about five percent of the weight to prespecified model D. Prespecified models A and B, the polyspline model, and the mean outcome across the Ys never received any weight in the final convex combination of models.

6 Bonus!

6.1 Try adding more algorithms to the `SuperLearner` library.

Below I try adding algorithms for penalized elastic net regression models and Bayesian generalized linear models.

```
# add penalized regression using elastic net and
# Bayes generalized linear model algorithms
sl_library_expanded <- c(sl_library, "SL.glmnet", "SL.bayesglm")
```

I will test these after I add my own wrapper functions to the library too!

6.2 Try writing your own wrapper function.

Below I write a function that predicts Y by taking its median across the observations.

```
# create an algorithm for the median of Y
SL.median <- function (Y, X, newX, family, obsWeights, id, ...)
{
    medianY <- median(Y)
    pred <- rep.int(medianY, times = nrow(newX))
    fit <- list(object = medianY)
    out <- list(pred = pred, fit = fit)
    class(out$fit) <- c("SL.median")
    return(out)
}
```

Below I write an algorithm that, for a continous outcome variable, creates pseudo-random values for Y within the range of the observed Y from a uniform distribution. For a binary outcome it generates a random value from a binomial distribution within the range of the observed Y.

```
# create an algorithm that creates random values for Y from a
# uniform distribution for Y within the range of the Y values for a
# "gaussian" family specification and and random values for Y up to the maximum
# value of Y from a binomial distribution for a "binomial" family specification
SL.random <- function (Y, X, newX, family, obsWeights, id, ...)
{
    if (family$family == "gaussian") {
        randomY <- runif(nrow(newX), min = min(Y), max = max(Y))
        pred <- randomY
        fit <- list(object = randomY)
        out <- list(pred = pred, fit = fit)
        class(out$fit) <- c("SL.random")
        return(out)
```

```
    }

    if (family$family == "binomial") {
        randomY <- rbinom(nrow(newX), size = max(Y), prob = 0.5)
        pred <- randomY
        fit <- list(object = randomY)
        out <- list(pred = pred, fit = fit)
        class(out$fit) <- c("SL.random")
        return(out)
    }
}
```

Now I run SuperLearner again using my new and exciting library with four new wrapper functions!

```
# add my functions to SuperLearner's library
sl_library_plus <- c(sl_library_expanded, "SL.median", "SL.random")

# run SuperLearner with my new not very exciting library
sl_out_plus <- SuperLearner(Y = obs_data$Y,
                            X = X,
                            SL.library = sl_library_plus,
                            cvControl = list(V = 20))

## Loading required namespace: arm
## Loading required namespace: glmnet

sl_out_plus

##
## Call:
## SuperLearner(Y = obs_data$Y, X = X, SL.library = sl_library_plus, cvControl = list(V = 20))
##
##
##
##                          Risk       Coef
## SL.glm.EstA_All      8.312903 0.00000000
## SL.glm.EstB_All     13.019920 0.00000000
## SL.glm.EstC_All      7.760012 0.21842813
## SL.glm.EstD_All     16.023770 0.02442038
## SL.ridge_All         3.994763 0.00000000
## SL.rpartPrune_All    4.702274 0.26058327
## SL.polymars_All     12.104627 0.04788184
## SL.mean_All        123.838484 0.00000000
## SL.glmnet_All        4.000507 0.00000000
## SL.bayesglm_All      3.994786 0.44868637
## SL.median_All      124.122580 0.00000000
## SL.random_All      345.749991 0.00000000
```

Among the pre-made wrapper functions I added, the Bayesian generalized linear modeling approach did very well and took nearly half the weight in the final ensemble. The ridge regression and penalized elastic net regression algorithms had almost the same risk but no weight in the final ensemble, likely due to collinearity between their results and those of the Bayes. The regression tree and model C algorithms still took a quarter and a fifth of the weight, respectively, followed by a few percentage points of weight for model D.

Among my own wrapper functions, it appears that ignoring the predictors entirely is a bad way to make predictions. But the risk of my little `SL.median` function was almost as low as the risk of the other function that ignored the predictors, the `SL.mean` function! It was almost not the worst! And my `SL.random` function did even worse than the worst. I'm somewhat reassured to see that the actual modeling attempts beat pseudo-random chance.

But . . . what if the pre-written function models are overfit so badly to these particular predictors that my median Y or pseudo-random chance models that completely *ignore* the predictors are actually better? Unlikely, but onward we march to `CV.SuperLearner` to make sure!

```
# run CV.SuperLearner with my expanded library
cv_sl_out_plus <- CV.SuperLearner(Y = obs_data$Y,
                                  X = X,
                                  SL.library = sl_library_plus,
                                  cvControl=list(V=5),
                                  innerCvControl=list(list(V=20)))

## Warning in CV.SuperLearner(Y = obs_data$Y, X = X, SL.library = sl_library_plus, : Only a single innerCvCont
is given, will be replicated across all cross-validation split calls to SuperLearner

# inspect the output
summary(cv_sl_out_plus)

##
## Call:
## CV.SuperLearner(Y = obs_data$Y, X = X, SL.library = sl_library_plus, 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   2.13534 0.045851 1.8472e+00   2.3096
##         Discrete SL   3.99288 0.073202 3.7644e+00   4.1793
##     SL.glm.EstA_All   8.31631 0.219051 7.7931e+00   8.6763
##     SL.glm.EstB_All  13.03150 0.241836 1.2611e+01  13.3201
##     SL.glm.EstC_All   7.77054 0.219580 7.3693e+00   8.1097
##     SL.glm.EstD_All  16.03115 0.268514 1.5752e+01  16.2287
##        SL.ridge_All   3.99261 0.073201 3.7644e+00   4.1793
##  SL.rpartPrune_All   5.06485 0.119220 4.6535e+00   5.2938
##     SL.polymars_All   0.51483 0.035733 9.3334e-03   2.5348
##         SL.mean_All 123.85873 1.469184 1.2075e+02 129.7246
##       SL.glmnet_All   3.99813 0.072934 3.7941e+00   4.1867
##     SL.bayesglm_All   3.99265 0.073184 3.7641e+00   4.1784
##       SL.median_All 124.25797 1.472435 1.2072e+02 130.7243
##       SL.random_All 342.01659 5.696803 3.2792e+02 353.6865

# returns the output for each call to Super Learner
cv_sl_out_plus$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     8.261987 0.00000000
## SL.glm.EstB_All    13.030257 0.00000000
## SL.glm.EstC_All     7.683230 0.20777281
## SL.glm.EstD_All    16.018401 0.03403678
## SL.ridge_All        3.949501 0.00000000
## SL.rpartPrune_All   4.940816 0.20168567
## SL.polymars_All     7.361034 0.14722050
## SL.mean_All       122.397321 0.00000000
## SL.glmnet_All       3.954970 0.00000000
## SL.bayesglm_All     3.949529 0.40928424
## SL.median_All     123.124863 0.00000000
## SL.random_All     336.627709 0.00000000
##
## $`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     8.224453 0.00000000
## SL.glm.EstB_All    12.984716 0.00000000
## SL.glm.EstC_All     7.761470 0.22230512
## SL.glm.EstD_All    16.109809 0.03330945
## SL.ridge_All        3.967849 0.00000000
## SL.rpartPrune_All   4.614641 0.25831117
## SL.polymars_All    13.901834 0.01922809
## SL.mean_All       124.467927 0.00000000
## SL.glmnet_All       3.973962 0.00000000
## SL.bayesglm_All     3.967830 0.46684617
## SL.median_All     124.806143 0.00000000
## SL.random_All     335.785572 0.00000000
##
## $`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     8.256134 0.00000000
## SL.glm.EstB_All    13.023757 0.00000000
## SL.glm.EstC_All     7.712062 0.22188748
## SL.glm.EstD_All    16.058995 0.03492089
## SL.ridge_All        3.994668 0.00000000
## SL.rpartPrune_All   4.996129 0.22431303
```

```
## SL.polymars_All    14.096316 0.06296375
## SL.mean_All       123.947450 0.00000000
## SL.glmnet_All       4.001497 0.00000000
## SL.bayesglm_All     3.994669 0.45591484
## SL.median_All     124.300862 0.00000000
## SL.random_All     338.925174 0.00000000
##
## $`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     8.449270 0.000000000
## SL.glm.EstB_All    13.131741 0.000000000
## SL.glm.EstC_All     7.807676 0.227055032
## SL.glm.EstD_All    16.028027 0.034411933
## SL.ridge_All        4.044249 0.000000000
## SL.rpartPrune_All   4.824691 0.255765679
## SL.polymars_All    16.905270 0.001245647
## SL.mean_All       124.571812 0.000000000
## SL.glmnet_All       4.051736 0.000000000
## SL.bayesglm_All     4.044243 0.481521708
## SL.median_All     124.712282 0.000000000
## SL.random_All     334.698621 0.000000000
##
## $`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     8.386427 0.000000000
## SL.glm.EstB_All    12.954082 0.000000000
## SL.glm.EstC_All     7.868713 0.231922241
## SL.glm.EstD_All    16.001335 0.045753346
## SL.ridge_All        4.058480 0.000000000
## SL.rpartPrune_All   5.022585 0.227917986
## SL.polymars_All    21.615022 0.005861249
## SL.mean_All       123.729351 0.000000000
## SL.glmnet_All       4.064688 0.000000000
## SL.bayesglm_All     4.058502 0.488545179
## SL.median_All     123.907282 0.000000000
## SL.random_All     329.916980 0.000000000

# condensed version of the output from CV.SL.out$AllSL with only the coefficients for each Super Learner run
```

```r
cv_sl_out_plus$coef

##   SL.glm.EstA_All SL.glm.EstB_All SL.glm.EstC_All SL.glm.EstD_All SL.ridge_All
## 1               0               0       0.2077728      0.03403678            0
## 2               0               0       0.2223051      0.03330945            0
## 3               0               0       0.2218875      0.03492089            0
## 4               0               0       0.2270550      0.03441193            0
## 5               0               0       0.2319222      0.04575335            0
##   SL.rpartPrune_All SL.polymars_All SL.mean_All SL.glmnet_All SL.bayesglm_All
## 1         0.2016857     0.147220500           0             0       0.4092842
## 2         0.2583112     0.019228090           0             0       0.4668462
## 3         0.2243130     0.062963753           0             0       0.4559148
## 4         0.2557657     0.001245647           0             0       0.4815217
## 5         0.2279180     0.005861249           0             0       0.4885452
##   SL.median_All SL.random_All
## 1             0             0
## 2             0             0
## 3             0             0
## 4             0             0
## 5             0             0

# returns the algorithm with lowest CV risk (discrete Super Learner) at each step.
cv_sl_out_plus$whichDiscrete

## $`1`
## [1] "SL.ridge_All"
##
## $`2`
## [1] "SL.bayesglm_All"
##
## $`3`
## [1] "SL.ridge_All"
##
## $`4`
## [1] "SL.bayesglm_All"
##
## $`5`
## [1] "SL.ridge_All"
```

The Super Learner algorithm beat all the discrete algorithms in average mean squared prediction error, predictably, followed by the Bayesian generalized linear model algorithm and then the ridge regression algorithm, which in some of the cross-validation folds beats the Bayes for the lowest risk. The final ensemble combination of models, however, only assigns weight to the Bayesian generalized linear model, which I suspect is due to collinearity between the results of the two algorithms. At the other end of candidate algorithm performance, my attempts at predciting without predictors had similarly awful risks after an additional layer of cross-validation and showed completely zeroed-out coefficients in both the overall final SuperLearner ensemble fit to all the data and in each of the five folds of the cross validation. Alas.