R Homework Three

Katherine Wolf
Introduction to Causal Inference (PH252D)
April 7, 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.

```r
library(tidyverse)

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

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

```r
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).

add explanation here

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$sin_W3 <- sin(obs_data$W3)
obs_data$W4_sq <- obs_data$W4 * obs_data$W4
obs_data$cos_W5 <- 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 + sin_W3 + W4_sq, data = train)
  model_b <- glm(Y ~ W1 + W2 + W4 + cos_W5, data = train)
  model_c <- glm(Y ~ W2 + W3 + W5 + W2:W5 + W4_sq + cos_W5, 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)
  cv_risk[V,3] <- mean((valid$Y - predict_c)^2)
  cv_risk[V,4] <- mean((valid$Y - predict_d)^2)
}
```

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

# 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

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

3.7 Fit the chosen algorithm on all the data.

```r
# 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        W4_sq       cos_W5
##    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
```

```r
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
## W4_sq       -10.80041    0.29801 -36.242   <2e-16 ***
## cos_W5        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?

I bet that we can, since we just arbitrarily picked four models and then stipulated that we had to choose one and only one of them.

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.

```r
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
```

```r
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`*.*
do this at the end

4.4 Create data frame X with the predictor variables.

Include the original predictor variables and the transformed variables.

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.

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?

do this at the end

5 Implement CV.SuperLearner.

5.1 Explain why we need CV.SuperLearner.

do this at the end

5.2 Run CV.SuperLearner.

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

6 Bonus!

6.1 Try adding more algorithms to the SuperLearner library.

do this at the end

6.2 Try writing your own wrapper function.

do this at the end