R Lab 3 - SuperLearner!

Introduction to Causal Inference

Goals:
1. Import a data set into `R`.
2. Code discrete SuperLearner (a.k.a. the cross-validation selector) to select the estimator with the lowest cross-validated risk from a library of candidate algorithms.
3. Use the `SuperLearner` package to build the best convex combination of algorithms.
4. Evaluate the performance of SuperLearner with `CV.SuperLearner`.

Next lab:
We will implement the inverse probability of treatment weighted (IPTW) estimator and explore the impact of positivity violations on estimator performance.

1 Background - Pirate Prediction!

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

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

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

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

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

2 Import the data set `RLab3.SuperLearner.csv`

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

   ```
   > ObsData<- read.csv("RLab3.SuperLearner.csv")
   ```

   For this to work, the `RLab3.SuperLearner.csv` file needs to be in your working directory, which can be accessed with the `getwd()` function. Alternatively, you can tell `R` the full path of the file. For example, if the `RLab3.SuperLearner.csv` file was downloaded to my desktop, I could use the following command:

   ```
   > ObsData<- read.csv("/Users/laura/Desktop/RLab3.SuperLearner.csv")
   ```

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

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

Solution:

```
> ObsData<- read.csv('RLab3.SuperLearner.csv')
> # get the column names
> names(ObsData)


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


> # show the obsv data on the first six ships
> head(ObsData)


  W1        W2 W3        W4 Y
1  0 4.0472206  2 0.9224837 0
2  0 4.8896172  1 0.8880435 0
3  0 4.2880436  1 0.7279743 0
4  0 0.5484213  0 0.9291807 0
5  0 2.4043513  1 0.1683894 0
6  0 4.4314941  1 0.3904279 1


> # recall: W1-map, W2-rum, W3-mutinies, W4-essential characteristics, Y- treasure
> summary(ObsData)


       W1              W2                W3              W4
 Min.   :0.00   Min.   :0.01048   Min.   :0.000   Min.   :0.001577
 1st Qu.:0.00   1st Qu.:1.20580   1st Qu.:0.000   1st Qu.:0.270563
 Median :0.00   Median :2.49865   Median :1.000   Median :0.515619
 Mean   :0.36   Mean   :2.46590   Mean   :1.016   Mean   :0.509001
 3rd Qu.:1.00   3rd Qu.:3.66682   3rd Qu.:2.000   3rd Qu.:0.749001
 Max.   :1.00   Max.   :4.99215   Max.   :5.000   Max.   :0.998503
       Y
 Min.   :0.000
 1st Qu.:0.000
 Median :0.000
 Mean   :0.247
 3rd Qu.:0.000
 Max.   :1.000
```

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

[1] 1000
```

3 The curse of dimensionality!

Suppose your Queen believes that the only predictors of a ship's success are possession of a treasure map ($W1$) and the number of mutinies in the last year ($W3$). Since these are categorical characteristics, she demands that you estimate the conditional probability of finding treasure, given these characteristics $E_0(Y|W1, W3)$, with the non-parametric maximum likelihood estimator (NPMLE).

1. **Use the `table` function to count the number of ships within strata of map possession $W1$ and mutinies $W3$.**
   Hint: Columns in the data frame `ObsData` can be accessed with the `$` operator.
2. **Are you wary of estimating the conditional expectation of Y with the sample proportion in each strata (i.e. the NPMLE)?** For example, try estimating the conditional probability of finding treasure without a map and with five counts of mutiny: $E_0(Y|W1 = 0, W3 = 5)$.
   Hint: use the logical `&`.

Solution:

```
> # 1) get the counts of ships within strata of map possession and mutinies
> table(ObsData$W1, ObsData$W3)

     0   1   2   3   4   5
  0 206 270 128  30   5   1
  1 110 151  75  22   2   0

> # row correspond to map possession and columns to # mutinies
>
> # 2) estimate the probability of finding treasure given no map and 5 counts of mutiny
> mean(ObsData$Y[ObsData$W1==0 & ObsData$W3==5])

[1] 1

> # The predicted probability of finding treasure without a map and with 5 counts of
> #  mutiny is 1 and based on a single observation.
>
> # the probability of finding treasure with a map & 5 counts of mutiny
> mean(ObsData$Y[ObsData$W1==1 & ObsData$W3==5])

[1] NaN
```

Arrrggg!!! The Curse of Dimensionality!!! Even with $n = 1,000$ ships, there are few ships with four or five counts of mutinies in the last year. The NPMLE, which is equivalent to fitting a fully saturated paramteric regression model, will break down with empty cells and may be an overfit with near-empty cells.

```
> # we can also try to fit a saturated parametric model using the factor function,
> # which tells R that W3 is a categorical variable
> head(factor(ObsData$W3))


[1] 2 1 1 0 1 1
Levels: 0 1 2 3 4 5


> glm(Y ~ W1*factor(W3), data=ObsData, family='binomial')


Call:  glm(formula = Y ~ W1 * factor(W3), family = "binomial", data = ObsData)

Coefficients:
   (Intercept)              W1     factor(W3)1     factor(W3)2     factor(W3)3
       -1.1914          0.3004         -0.1949         -0.2749         -0.4180
   factor(W3)4     factor(W3)5  W1:factor(W3)1  W1:factor(W3)2  W1:factor(W3)3
      -14.3747         16.7575          0.3828          0.5906          0.3282
W1:factor(W3)4  W1:factor(W3)5
       -0.3004              NA

Degrees of Freedom: 999 Total (i.e. Null);  989 Residual
Null Deviance:            1118
Residual Deviance: 1093         AIC: 1115
```

Take-home message: In many data applications, our statistical model $\mathcal{M}$ is non-parameteric, but the NPMLE is not well-defined. There can be strata with zero or only a few observations. In these situations, estimation of the conditional mean function with the NPMLE (i.e. a fully saturated parametric regression model) will be an overfit. Thereby, we need an alternative estimation approach. We could describe the probability of finding treasure given predictor variables with a lower dimensional parametric model. However, we often do not know enough to *a priori*-specify the correct parametric model for $E_0(Y|W1, W2, W3, W4)$. If the specified parametric model is incorrect, our point estimates and inference will be biased. Trying a bunch of parametric regressions, looking at the results, fiddling with the regrssion specifications, and choosing the "best"(using arbitrary criteria) also leads to biased point estimates and misleading inference. Therefore, we are going to use discrete SuperLearner to choose between a library of *a priori*-specified candidate prediction models.

4 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 binary, we can use the L2 loss function or the negative log likelihood loss:

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

For the purposes of this lab, we are going to use the L2 loss function as our measure of performance. The expectation of the loss function under the true data generating distribution P_0 is called the *risk*.

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

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

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

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

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

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

3. **Split the data into $V = 10$ folds.** With $n = 1000$ observations total, we want $n/10 = 100$ in each fold. For simplicity let us define the first hundred observations to be the first fold, the second hundred to be the second fold, and so forth.

   (a) **Create the vector `Fold` by with the `c()` and `rep()` functions.**

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

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

   (b) **Add the `Fold` vector to the data frame `ObsData` with the `data.frame` function.**

4. **Create an empty matrix `CV.risk` with 10 rows and 4 columns to hold the cross-validated risk for each algorithm, evaluated at each fold.**

   Hint: the following code creates a matrix `temp` of `NA` of size 2 by 2:

```
> temp<- matrix(NA, nrow=2, ncol=2)
> temp
```

```
     [,1] [,2]
[1,]   NA   NA
[2,]   NA   NA
```

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

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

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

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

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

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

(f) **Estimate the cross-validated risk estimate for each algorithm based on the L2 loss function.** Take the `mean` of the squared difference between the outcomes Y in the validation set and the predicted probabilities. **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.** Apply the `colMeans` function to the matrix `CV.risk`.

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

8. **Can we do better?**

Solution:

1. Squawk! Polley, the parrot, knows the answer.

```
> # 2. Transformed variable
> W2sq <- ObsData$W2*ObsData$W2
> sinW2sq<- sin(W2sq)
> logW2<- log(ObsData$W2)
> W4sq <- ObsData$W4*ObsData$W4
> sinW4sq <- sin(W4sq)
> logW4 <- log(ObsData$W4)


> ObsData<- data.frame(ObsData, W2sq, sinW2sq, logW2, W4sq, sinW4sq, logW4)


> ################
> # 3. Split the data into V = 10 folds
> # create the vector Fold
> Fold<- c(rep(1, 100), rep(2, 100), rep(3, 100),rep(4, 100),rep(5, 100),
+   rep(6, 100),rep(7, 100), rep(8, 100), rep(9, 100), rep(10, 100))
> # add the fold vector to the data frame
> ObsData<- data.frame(ObsData, Fold)
> head(ObsData)
```

```
  W1        W2 W3        W4 Y       W2sq    sinW2sq      logW2       W4sq
1  0 4.0472206  2 0.9224837 0 16.3799949 -0.6225771  1.3980304 0.85097615
2  0 4.8896172  1 0.8880435 0 23.9083568 -0.9405970  1.5871140 0.78862131
3  0 4.2880436  1 0.7279743 0 18.3873183 -0.4459520  1.4558306 0.52994653
4  0 0.5484213  0 0.9291807 0  0.3007659  0.2962518 -0.6007115 0.86337678
5  0 2.4043513  1 0.1683894 0  5.7809051 -0.4814253  0.8772801 0.02835499
6  0 4.4314941  1 0.3904279 1 19.6381399  0.7093559  1.4887368 0.15243396
     sinW4sq       logW4 Fold
1 0.75192429 -0.08068559    1
2 0.70938221 -0.11873452    1
3 0.50548721 -0.31748958    1
4 0.76004138 -0.07345205    1
5 0.02835119 -1.78147607    1
6 0.15184432 -0.94051191    1


> # 4 create a matrix CV.risk of size 10 by 4.
> CV.risk<- matrix(NA, nrow=10, ncol=4)


> # 5. Implementing discrete Superlearner
> # use a for loop to fit each estimator on the training set,
> # predict the prob of finding treasure for the validation set,
> # evaluate the cross-validated risk....
>
> # a. the for loop runs from V=1 to V=10
> for(V in 1:10){
+
+   # b. create the validation set by finding the ships (rows) in Fold==V and
+   # grabbing all the data (columns)
+   valid<- ObsData[Fold==V, ]
+
+   # c.create the training set by finding the ships (rows) in Fold!=V and
+   # grabbing all the data (columns)
+   train<- ObsData[Fold !=V, ]
+
+   #sanity check when building
+   #nrow(valid) - should be 100; nrow(train) - should be 900
+
+   # d. fit each algorithm on the training set -
+   # be sure to specify family= binomial for the logistic function
+   # be sure to specify the data as the training set
+
+   EstA<- glm(Y~ W1*W3 + W4sq, family='binomial', data=train)
+   EstB<- glm(Y~ W1+ logW2 + W3*W4, family='binomial', data=train)
+   EstC<- glm(Y~ W1*W2*W4, family='binomial', data=train)
+   EstD<- glm(Y~ W1*sinW2sq+ logW4, family='binomial', data=train)
+
+   # Note: we are specifying the model formula shorthand (lazy)
+   # For example, the equivalent command for EstA would be
+   # glm(Y~ W1 + W3 + W1*W3 + W4sq, family='binomial', data=train)
+
+   # e) for each algorithm obtain the predicted probability for each ship
+   #  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 predicted prob and the outcomes
+   #   for the validation set
+   # head(data.frame(PredA, PredB, PredC, PredD, valid$Y))
+
+   # f) estimate the cross-validated risk for each algorithm
+   # This uses the L2 loss... this is the average squared difference between the
+   # outcomes in the validation set and the predicted probability based on
+   # the estimator fit with the training set.
+   CV.risk[V,]<- c(mean((valid$Y - PredA)^2), mean((valid$Y - PredB)^2),
+     mean((valid$Y - PredC)^2), mean((valid$Y - PredD)^2))
+
+   # If we had specified our measure of performance to be the negative log
+   # likelihood loss. The following code would estimate the cross-validated risk
+   # based on this loss function.
+   # CV.risk[V,] <- c( -mean(valid$Y*log(PredA)+ (1-valid$Y)*(log(1-PredA))),
+   #  - mean(valid$Y*log(PredB)+ (1-valid$Y)*(log(1-PredB))),
+   #  - mean(valid$Y*log(PredC)+ (1-valid$Y)*(log(1-PredC))),
+   #  - mean(valid$Y*log(PredD)+ (1-valid$Y)*(log(1-PredD))))
+ }


> # 6. Algorithm with lowest CV risk
> colMeans(CV.risk)


[1] 0.1818298 0.1780452 0.1647564 0.1556738


> # 8. select the algorithm with the lowest cross-validated risk
> # run the algorithm on  the all the data to get the prediction function
> EstD.all<- glm(Y~ W1*sinW2sq+ logW4, family='binomial', data=ObsData)
> EstD.all


Call:  glm(formula = Y ~ W1 * sinW2sq + logW4, family = "binomial",
    data = ObsData)

Coefficients:
(Intercept)           W1      sinW2sq        logW4   W1:sinW2sq
    -0.7974       0.7175      -0.2235       0.8982       2.0269

Degrees of Freedom: 999 Total (i.e. Null);  995 Residual
Null Deviance:            1118
Residual Deviance: 970         AIC: 980
```

8. The discrete Super Learner can only do as well as the best algorithm in our library (here, four logistic regression models). We can potentially improve its performance by expanding the library with new algorithms. We can also use the `SuperLearner` package to expand our library to include convex combinations of algorithms and potentially obtain a better predictor of finding treasure given the measured covariates.

Note: coding the full SuperLearner to get the best weighted combination of algorithms is beyond the scope of this class, but feel free to try.

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

1. **If you have not done so already, install the `SuperLearner` package.** Click on the `Packages & Data` tab. Select `Package Installer`. Search for `SuperLearner`. Check the box for `Install Dependencies` and click `Install Selected`.

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

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

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

   ```
   > SL.step
   ```

5. **Use the `source` function to load script file `RLab3.SuperLearner.Wrappers.R`, which includes code for the wrapper functions for the *a priori*-specified parametric estimators.**

   ```
   > source('RLab3.SuperLearner.Wrappers.R')
   ```

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

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

   Note: We are choosing these simple algorithms in the interest of time. Remember Anna's Boy Band heuristic; there are a lot more fun and exciting algorithms out there. Feel free to try out others.

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

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

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

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

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

Solution:

```
> # 2 load SuperLearner
> library('SuperLearner')


> # 3 check out the SuperLearner function
> ?SuperLearner


> # 4. Default avaliable wrappers can be accessed with
> listWrappers()


 [1] "SL.bart"             "SL.bayesglm"         "SL.caret"
 [4] "SL.caret.rpart"      "SL.cforest"          "SL.earth"
 [7] "SL.gam"              "SL.gbm"              "SL.glm"
[10] "SL.glm.interaction"  "SL.glmnet"           "SL.ipredbagg"
[13] "SL.knn"              "SL.leekasso"         "SL.loess"
[16] "SL.logreg"           "SL.mean"             "SL.nnet"
[19] "SL.polymars"         "SL.randomForest"     "SL.ridge"
[22] "SL.rpart"            "SL.rpartPrune"       "SL.step"
[25] "SL.step.forward"     "SL.step.interaction" "SL.stepAIC"
[28] "SL.svm"              "SL.template"
[1] "All"
[1] "screen.corP"           "screen.corRank"        "screen.glmnet"
[4] "screen.randomForest"   "screen.SIS"           "screen.template"
[7] "screen.ttest"          "write.screen.template"


> # SL.step is the wrapper function for fitting generalized linear models (ie
> # running glm on all predictors) and then using  AIC to chose the best model
> # in a stepwise algorithm .
> SL.step


function (Y, X, newX, family, direction = "both", trace = 0,
    k = 2, ...)
{
    fit.glm <- glm(Y ~ ., data = X, family = family)
    fit.step <- step(fit.glm, direction = direction, trace = trace,
        k = k)
    pred <- predict(fit.step, newdata = newX, type = "response")
    fit <- list(object = fit.step)
    out <- list(pred = pred, fit = fit)
    class(out$fit) <- c("SL.step")
    return(out)
}
<environment: namespace:SuperLearner>


> # many other wrappers can be written.


> # 5. Load the wrapper code for the 4 algorithms
> source('RLab3.SuperLearner.Wrappers.R')


> # 6. let's create a library of our above algorithms
> SL.library<- c('SL.glm.EstA', 'SL.glm.EstB', 'SL.glm.EstC', 'SL.glm.EstD')
```

```
> # 7. data frame X of predictor variables
> # here, we are copying the column vectors of the predictor variables into X
> X<- subset(ObsData, select= -Y )
> tail(X)


     W1        W2 W3        W4       W2sq    sinW2sq      logW2       W4sq
995   1 0.7268167  2 0.2617021  0.5282625  0.5040334 -0.3190810 0.06848799
996   0 2.0395120  0 0.7690135  4.1596092 -0.8510683  0.7127106 0.59138180
997   1 3.3193944  1 0.4918185 11.0183791 -0.9997400  1.1997824 0.24188543
998   0 3.6907031  1 0.7389013 13.6212896  0.8698603  1.3058170 0.54597506
999   0 3.5970363  1 0.6667441 12.9386701  0.3637583  1.2801103 0.44454768
1000  0 4.0956483  0 0.9210684 16.7743350 -0.8754527  1.4099250 0.84836701
        sinW4sq       logW4 Fold
995  0.06843446 -1.34054842   10
996  0.55750868 -0.26264672   10
997  0.23953359 -0.70964555   10
998  0.51925165 -0.30259099   10
999  0.43004957 -0.40534898   10
1000 0.75020166 -0.08222097   10


> # 8. Call Superlearner
> # ObsData$Y is the vector of outcomes.
> SL.out<- SuperLearner(Y=ObsData$Y, X=X, SL.library=SL.library, family='binomial',
+     cvControl=list(V=10))


> # 9. examine the SL.out object
> SL.out


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


                     Risk     Coef
SL.glm.EstA_All 0.1816773 0.000000
SL.glm.EstB_All 0.1775385 0.000000
SL.glm.EstC_All 0.1651437 0.365651
SL.glm.EstD_All 0.1564785 0.634349
```

The `Risk` column gives the cross-validated risk for each algorithm averaged across the 10 folds. The `Coef` column gives the weight of each algorithm in the convex combination. The algorithm with the lowest average cross-validated risk was Estimator D. It was also given the most weight when building the best combination of prediction algorithms. Nonetheless, the weight given to Estimator C is not trivial.

As many of you noticed, running `SuperLearner` multiple times may result in slightly different risk estimates and weights. The package assigns the folds randomly (whereas we set the first 100 observations to be in Fold 1, the second 100 to Fold 2, and so forth). For more stability, we could increase the number of folds. The next section `CV.SuperLearner` is dedicated to evaluating the performance of `SuperLearner`.

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

```
[1] 0.1818298 0.1780452 0.1647564 0.1556738


> # the CV risks are close (within roundoff error). 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"        "varNames"          "validRows"
[13] "method"            "whichScreen"       "control"
[16] "cvControl"         "errorsInCVLibrary" "errorsInLibrary"


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

6 Evaluate the performance of SuperLearner.

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

   ```
   > ?CV.SuperLearner
   ```

2. **Evaluate the performance of the SuperLearner algorithm by running** `CV.SuperLearner`**.** Again be sure to specify the outcome `Y`, predictors `X`, `family`, `SL.library`, `V=10` and `cvControl=list(V=10)`. This might take a couple minutes.

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

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

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

Solution:

```
> ?CV.SuperLearner


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


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

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

Risk is based on: Mean Squared Error

All risk estimates are based on V =  10

       Algorithm     Ave        se     Min     Max
   Super Learner 0.15308 0.0067959 0.11173 0.19127
     Discrete SL 0.15607 0.0070604 0.11443 0.19651
 SL.glm.EstA_All 0.18184 0.0069459 0.15451 0.21982
 SL.glm.EstB_All 0.17814 0.0069680 0.15219 0.21688
 SL.glm.EstC_All 0.16549 0.0073524 0.12281 0.20596
 SL.glm.EstD_All 0.15607 0.0070604 0.11443 0.19651
```

`CV.SuperLearner` evaluates the performance of `SuperLearner` by adding an extra layer of cross-validation. This provides a check against overfitting and allows us to compare its performance to other algorithms. When evaluating the performance of `SuperLearner`, we want to use data that it did not see when building the prediction function.

`Super Learner`, using the optimal combination of algorithms, had the lowest "honest" cross-validated risk. The discrete cross-validation selector (`Discrete SL`) corresponding to Estimator D (`SL.glm.EstD`) had the second lowest "honest" cross-validated risks. These risk estimates were computed running the full SuperLearning algorithm on $9/10^{th}$ of the data, and evaluating the risk on the remaining $1/10^{th}$ of the data and repeating across all folds.

```
> # Get the output for each call to SuperLearner. We see that the cross-validated risks
> # and coefficients fluctuation a little bit.
> CV.SL.out$AllSL
```

Solution:

7 Appendix

```
> # This is the code used to generate data set RLab3.SuperLearner.csv
> set.seed(1216)
> n=1000
> W4 <- runif(n, 0, 1) # parrot, peg leg, eye patch
> W2 <- runif(n, 0, 5) # rum
> W4sq<- W4*W4
> W2sq<- W2*W2
> W1 <- rbinom(n, size=1, prob=plogis(W4sq - 4*W4+1)) #treasure map
> W3 <- rbinom(n, size=5, prob=0.2) # mutinies
> Y<- rbinom(n, size=1, prob= plogis(-1 + 3*W1 - sin(W4sq) +3*W1*sin(W2sq) + 3*W1*log(W4) ))
> ObsData<- data.frame(W1,W2, W3, W4, Y)
> summary(ObsData)
> write.csv(ObsData, file='RLab3.SuperLearner.csv', row.names=F)
```

```
> #rm(list=ls())
>
> # So the true mean Y given covariates W is a logistic function of the possession of
> # a treasure map (W1), rum (W2) and the proportion of pirates with parrots, peg legs
> # or eye patches (W4). The number of mutinies (W3) does not matter.
```