R Lab 4 - Super Learner!

Laura B. Balzer

Biostat 683 - Intro. to Causal Inference

Goals:
1. Code Discrete Super Learner (a.k.a., the cross-validation selector) to select the estimator with the lowest cross-validated risk from a library of candidate algorithms.
2. Use the `SuperLearner` package to build the best convex combination of predictions from the candidate algorithms.
3. Evaluate the performance of Super Learner with `CV.SuperLearner`.

Next lab:
We will implement targeted minimum loss-based estimation (TMLE).

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 `RLab4.SuperLearner.csv`

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

   ```
   > ObsData<- read.csv("RLab4.SuperLearner.csv")
   ```

   For this to work, the `RLab4.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 `RLab4.SuperLearner.csv` file was downloaded to my desktop, I could use the following command:

   ```
   > ObsData<- read.csv("/Users/laura/Desktop/RLab4.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('RLab4.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, $\mathbb{E}_0(Y|W1, W3) = \mathbb{P}_0(Y = 1 \mid W1, W3)$ 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 predicting success via 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: $\mathbb{E}_0(Y|W1 = 0, W3 = 5) = \mathbb{P}_0(Y = 1 \mid W1 = 0, W3 = 5)$.

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


> # rows 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 parametric regression model, will break down with empty cells and may be an over-fit 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 (i.e., a fully stratified approach) is not well-defined. There can be strata with zero or only a few observations. Thereby, we need an alternative estimation approach. We could use a lower dimensional parametric model to describe the probability of finding treasure given predictor variables. However, we often do not know enough to *a priori*-specify the correct regression. If the specified parametric regression is incorrect, our point estimates and inference may be biased. Trying a bunch of regressions, looking at the results, fiddling with the specifications, and choosing the "best" (using arbitrary criteria) also leads to biased point estimates and misleading inference. Therefore, we are going to use Discrete Super Learner to choose between a library of *a priori*-specified candidate prediction algorithms, using cross-validation, and according to our selected loss function.

Reminder: This discussion also applies to estimation of the conditional mean outcome $\mathbb{E}_0(Y|A,W)$ used in a substitution estimator (a.k.a., G-computation estimator) and for the propensity score $\mathbb{P}_0(A = 1|W)$ used in IPTW.

4 Code Discrete Super Learner to select the estimator with the smallest cross-validated risk estimate

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

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

Since the outcome is binary, we could 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 $\mathbb{P}_0$ is called the *risk*.

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

$$\bar{Q}^a(W) = logit^{-1}\big[\beta_0 + \beta_1 W1 + \beta_2 W3 + \beta_3 W1^* W3 + \beta_5 W4^2\big]$$
$$\bar{Q}^b(W) = logit^{-1}\big[\beta_0 + \beta_1 W1 + \beta_2 log(W2) + \beta_3 W3 + \beta_4 W4 + \beta_5 W3^* W4\big]$$
$$\bar{Q}^c(W) = logit^{-1}\big[\beta_0 + \beta_1 W1 + \beta_2 W2 + \beta_3 W4 + \beta_4 W1^* W2 + \beta_5 W1^* W4 + \beta_6 W2^* W4 + \beta_7 W1^* W2^* W4\big]$$
$$\bar{Q}^d(W) = logit^{-1}\big[\beta_0 + \beta_1 W1 + \beta_2 sin(W2^2) + \beta_3 W1^* sin(W2^2) + \beta_4 log(W4)\big]$$

where $W = (W1, W2, W3, W4)$. Therefore, our library consists of four parametric regression, denoted with the superscripts $a - d$. Using cross-validation, we can generate an "honest" estimate of risk for each candidate $\bar{Q}(W) = \mathbb{E}(Y|W)$. 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.

Caveat: In most real-data applications, we would not have the knowledge to *a priori*-specify transformations using trig functions. This is for illustration of how we could encode subject matter knowledge into our parametric regressions. (For a real example, see https://academic.oup.com/cid/article/71/9/2326/5614347 where we included pre-specified interactive terms to improve our Super Learner.) The corresponding homework will expand our library to include both expert-specified, parametric approaches as well as non-parametric machine learning algorithms.

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

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

```
> 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.
Hint: We can 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, you could use the `sample` function (without replacement) to get 10 folds of size 100.

4. **Create a matrix `Pred` with 1000 rows for the ships and 4 columns to hold the cross-validated predictions for each observation according to each candidate algorithm.**

```
> Pred <- matrix(NA, nrow=1000, ncol=4)
```

5. **Create a matrix `CV.risk` with 10 rows for the folds and 4 columns to hold the cross-validated risk estimates for each algorithm.**

6. **To implement Discrete Super Learner, 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 observations 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) **Save the cross-validated predictions for each ship in the validation set at the appropriate row in the matrix `Pred`.**

   (g) **Estimate the cross-validated risk estimate for each algorithm based on the L2 loss function.** Take the average of the squared difference between the outcomes Y in the validation set and their cross-validated predicted probabilities. **Assign the cross-validated risks as a row in the data frame `CV.risk`.**

7. **Select the algorithm with the lowest average cross-validated risk across the folds.** Apply the `colMeans` function to the matrix `CV.risk`.

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

9. **How can we come up with an even better prediction function than the one selected?**

10. See the Bonus for hand-coding of the ensemble Super Learner.

Solution:

1. Squawk! Polley, the parrot, knows the answer. (This is a joke; please laugh now... Eric Polley is one of the authors and the maintainer of the `SuperLearner` package.)

```
> # 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))


> # 4. create a matrix to hold the cross-validated predictions
> Pred <- matrix(NA, nrow=1000, ncol=4)


> # 5 create a matrix CV.risk of size 10 by 4.
> CV.risk<- data.frame(matrix(NA, nrow=10, ncol=4) )
> # label the columns for the candidate estimators
> colnames(CV.risk)<- c('EstA', 'EstB', 'EstC', 'EstD')


> # 6. Implementing Discrete Super Learner
> # 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')
+
+   # f) save the cross-validated predictions
+   Pred[Fold==V, ] <- cbind(PredA, PredB, PredC, PredD)
+
+   # can see the difference between the predicted prob and the outcomes
+   #   for the validation set
+   # head(data.frame(Pred[Fold==V,], Y=valid$Y))
+
+   # g) estimate the cross-validated risk for each algorithm
+   # This uses the L2 loss... this is the average squared difference between the
+   # outcomes in the validation set and the predicted probability based on
+   # the estimator fit with the training set.
+   CV.risk[V,]<- c(mean((valid$Y - PredA)^2), mean((valid$Y - PredB)^2),
+     mean((valid$Y - PredC)^2), mean((valid$Y - PredD)^2))
+
+   # We could equivalently code as
+   # colMeans( (valid$Y - Pred[Fold==V,])^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,] <- -colMeans( valid$Y*log(Pred[Fold==V,])+
+   #                   (1-valid$Y)*(log(1-Pred[Fold==V,]) ) )
+ }

> # 7. Algorithm with lowest CV risk
> colMeans(CV.risk)

     EstA      EstB      EstC      EstD
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

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

1. **If you have not done so already, install the `SuperLearner` package.** In RStudio, click on `Tools` in the menu and select `Install Packages...` Search for `SuperLearner`. Check the box for `Install Dependencies` and click `Install`.

2. **Load the Super Learner 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 `RLab4.SuperLearner.Wrappers.R`, which includes code for the wrapper functions for the *a priori*-specified parametric estimators.**
```
> source('RLab4.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 our Spice Girls principle; 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. **Set the seed to 1.**

9. **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))
```

10. **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.bartMachine"      "SL.bayesglm"         "SL.biglasso"
 [4] "SL.caret"            "SL.caret.rpart"      "SL.cforest"
 [7] "SL.earth"            "SL.extraTrees"       "SL.gam"
[10] "SL.gbm"              "SL.glm"              "SL.glm.interaction"
[13] "SL.glmnet"           "SL.ipredbagg"        "SL.kernelKnn"
[16] "SL.knn"              "SL.ksvm"             "SL.lda"
[19] "SL.leekasso"         "SL.lm"               "SL.loess"
[22] "SL.logreg"           "SL.mean"             "SL.nnet"
[25] "SL.nnls"             "SL.polymars"         "SL.qda"
[28] "SL.randomForest"     "SL.ranger"           "SL.ridge"
[31] "SL.rpart"            "SL.rpartPrune"       "SL.speedglm"
[34] "SL.speedlm"          "SL.step"             "SL.step.forward"
[37] "SL.step.interaction" "SL.stepAIC"          "SL.svm"
[40] "SL.template"         "SL.xgboost"
[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)
}
<bytecode: 0x7f80bc503b50>
<environment: namespace:SuperLearner>

> # many other wrappers can be written.

> # 5. Load the wrapper code for the 4 algorithms
> source('RLab4.SuperLearner.Wrappers.R')

> # 6. Set the seed.
> set.seed(1)
> # 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
995  0.06843446 -1.34054842
996  0.55750868 -0.26264672
997  0.23953359 -0.70964555
998  0.51925165 -0.30259099
999  0.43004957 -0.40534898
1000 0.75020166 -0.08222097


> # 8. Call Super Learner
> # 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 0.1825828 0.0000000
SL.glm.EstB_All 0.1786306 0.0000000
SL.glm.EstC_All 0.1655594 0.3512984
SL.glm.EstD_All 0.1561779 0.6487016
```

The `Risk` column gives the cross-validated risk estimate 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 you may have 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)


     EstA      EstB      EstC      EstD
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"        "cvFitLibrary"      "varNames"
[13] "validRows"         "method"            "whichScreen"
[16] "control"           "cvControl"         "errorsInCVLibrary"
[19] "errorsInLibrary"   "metaOptimizer"     "env"
[22] "times"


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

6 Evaluate the performance of Super Learner.

For a "honest" measure of performance, we add another layer of cross-validation to Super Learner.

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

   ```
   > ?CV.SuperLearner
   ```

2. **Evaluate the performance of the Super Learner algorithm by running `CV.SuperLearner`.** Again be sure to specify the outcome `Y`, predictors `X`, `family`, and `SL.library`. (Here, we are using the default settings for the cross-validation splits.) This might take a couple minutes to run.

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

   This function is partitioning the data into V*=10 folds, running the whole Super Learner 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')

> summary(CV.SL.out)

Call:
CV.SuperLearner(Y = ObsData$Y, X = X, family = "binomial", SL.library = SL.library)


Risk is based on: Mean Squared Error

All risk estimates are based on V =  10

       Algorithm     Ave        se     Min     Max
   Super Learner 0.15316 0.0067650 0.13550 0.17336
     Discrete SL 0.15641 0.0070544 0.13561 0.17477
 SL.glm.EstA_All 0.18213 0.0069461 0.14623 0.20328
 SL.glm.EstB_All 0.17801 0.0069529 0.14262 0.20084
 SL.glm.EstC_All 0.16520 0.0073292 0.12695 0.19792
 SL.glm.EstD_All 0.15641 0.0070544 0.13561 0.17477
```

`CV.SuperLearner` evaluates the performance of `SuperLearner` by adding an extra layer of cross-validation. This provides a check against over-fitting 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 estimate. The discrete cross-validation selector (`Discrete SL`) corresponding to Estimator D (`SL.glm.EstD`) had the second lowest “honest” cross-validated risk estimate. These risk estimates were computed running the full Super Learner 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 Super Learner. We see that the cross-validated risks
> # and coefficients fluctuation a little bit.
> CV.SL.out$AllSL

$`1`

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



                     Risk      Coef
SL.glm.EstA_All 0.1826170 0.0000000
SL.glm.EstB_All 0.1788710 0.0000000
SL.glm.EstC_All 0.1650220 0.3707679
SL.glm.EstD_All 0.1571378 0.6292321

$`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 0.1849561 0.0000000
SL.glm.EstB_All 0.1816013 0.0000000
SL.glm.EstC_All 0.1688504 0.3202498
SL.glm.EstD_All 0.1571532 0.6797502

$`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 0.1862509 0.0000000
SL.glm.EstB_All 0.1826425 0.0000000
SL.glm.EstC_All 0.1691052 0.3314292
SL.glm.EstD_All 0.1571661 0.6685708

$`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 0.1811205 0.0000000
SL.glm.EstB_All 0.1771743 0.0000000
SL.glm.EstC_All 0.1650585 0.3516178
SL.glm.EstD_All 0.1560632 0.6483822

$`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 0.1821060 0.0000000
SL.glm.EstB_All 0.1789335 0.0000000
SL.glm.EstC_All 0.1669091 0.3409911
SL.glm.EstD_All 0.1572863 0.6590089

$`6`

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 0.1795899 0.0000000
SL.glm.EstB_All 0.1761932 0.0000000
SL.glm.EstC_All 0.1627031 0.3786659
SL.glm.EstD_All 0.1544018 0.6213341

$`7`

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 0.1803049 0.0000000
SL.glm.EstB_All 0.1760451 0.0000000
SL.glm.EstC_All 0.1631481 0.3694579
SL.glm.EstD_All 0.1549119 0.6305421

$`8`

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 0.1835089 0.0000000
SL.glm.EstB_All 0.1795756 0.0000000
SL.glm.EstC_All 0.1664795 0.3730602
SL.glm.EstD_All 0.1586463 0.6269398

$`9`

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 0.1812915 0.0000000
SL.glm.EstB_All 0.1773226 0.0000000
SL.glm.EstC_All 0.1618044 0.3830489
SL.glm.EstD_All 0.1542813 0.6169511

$`10`

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 0.1809579 0.0000000
SL.glm.EstB_All 0.1779846 0.0000000
SL.glm.EstC_All 0.1652105 0.3553639
SL.glm.EstD_All 0.1555183 0.6446361
```

7 Bonus: Coding the weights

Try the following code to create the weights from your hand-coded Super Learner when the goal is to minimize the expected L2 loss function. (If you have a different loss function, it will be a different procedure to find the optimal weights.)

```
> # Load in the nnls library
> library(nnls)
> # Create a new data frame with the observed outcome (Y) and CV-predictions from the 4 algorithms
> X<- cbind(ObsData$Y, Pred)
> head(X)
```

```
     [,1]      [,2]      [,3]      [,4]       [,5]
[1,]    0 0.2517990 0.2652065 0.2083629 0.34108956
[2,]    0 0.2644518 0.2482541 0.2180554 0.35259727
[3,]    0 0.2232686 0.2113653 0.2097259 0.28410396
[4,]    0 0.3018996 0.3335361 0.1666539 0.28909303
[5,]    0 0.1570145 0.1164404 0.2406333 0.09424833
[6,]    1 0.1717494 0.1407970 0.2055013 0.14066346
```

```
> ## 4: estimate weights using non-linear least squares
> weights <- nnls(X[,2:5], X[,1])$x
> # then normalize to sum to 1 by dividing by the sum of the weights
> alpha<-as.matrix(weights/sum(weights))
> round(alpha,3)

      [,1]
[1,] 0.000
[2,] 0.000
[3,] 0.359
[4,] 0.641

> # compare to the package's coefficients
> 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 0.1825828 0.0000000
SL.glm.EstB_All 0.1786306 0.0000000
SL.glm.EstC_All 0.1655594 0.3512984
SL.glm.EstD_All 0.1561779 0.6487016

> #----------------
> ## fit all algorithms to original data & generate predictions
> PredA<- predict(glm(Y~ W1*W3 + W4sq, family='binomial',
+                     data=ObsData), type='response')
> PredB<- predict(glm(Y~ W1+ logW2 + W3*W4, family='binomial',
+                     data=ObsData), type='response')
> PredC<- predict(glm(Y~ W1*W2*W4, family='binomial',
+                     data=ObsData), type='response')
> PredD<- predict(glm(Y~ W1*sinW2sq+ logW4, family='binomial',
+                     data=ObsData), type='response')
> Pred <- cbind(PredA, PredB, PredC, PredD)
> # Take a weighted combination of predictions using nnls coeficients as weights
> Y.SL <- Pred%*%alpha
> # comparing the (non-cross-validated) weighted predictions between
> # our our hand-coded ensemble SuperLearner and the package
> mean( (ObsData$Y- Y.SL)^2)

[1] 0.1508343

> mean( (ObsData$Y- SL.out$SL.predict)^2)

[1] 0.1508343
```

Solution:

8 Appendix

```
> # This is the code used to generate data set RLab4.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='RLab4.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.
```