Introduction to causal inference with machine learning

Lab 1: Cross-validation and super learning

David Benkeser
Emory University

Iván Díaz
Weill Cornell Medical College

6th Seattle Symposium in Biostatistics

November 14, 2020

Contents of this Lab

- Understanding cross-validation and ensemble estimators.
- Understanding and executing basic calls to `SuperLearner`.
- Writing your regression estimator that is compatible with `SuperLearner`.
- Writing your own variable screening wrapper that is compatible with `SuperLearner`.
- Evaluating the performance of `SuperLearner`.
- Writing up the results of a super learner analysis.

Installing R packages

To follow along with this demonstration, you will need several R packages.

- Packages are freely available from the Comprehensive R Archive Network (CRAN).
- Packages are downloaded to your local computer via the `install.packages` function.
- Packages are loaded into your current R session via ‘require‘ or ‘library‘

```r
# these packages are needed to execute the demo
pkgs <- c("SuperLearner", "quadprog", "nloptr", "MASS", "ggplot2", "nnls")
# see what packages are currently installed
installed_pacakges <- row.names(installed.packages())
# loop over the needed packages
for(p in pkgs){
  # check if package is installed
  already_installed <- p %in% installed_pacakges
  # if not already installed, install it
  if(!already_installed){
    install.packages(p)
  }
  # and load package
  library(p, character.only = TRUE)
}
```

Simulating data

We will use simulated data sets where we know the truth to demonstrate some key ideas.

- We generate a data set of $n = 300$ observations with two covariates.
- Treatment probability is $g(W) = \text{expit}(-1 + W_1 - W_2 + W_1 W_2)$.
- Conditional distribution of Y is Normal with mean $\bar{Q}(A, W) = W_1 - W_2 + A$.

```r
# set a seed for reproducibility
set.seed(212)
# sample size
n <- 100
# W1 has Normal distribution, W2 has Uniform distribution
W1 <- rnorm(n = n); W2 <- runif(n = n)
# pr(A = 1 | W) is logistic linear in W
g1W <- plogis(-1 + W1 - W2 + W1*W2)
# generate binary treatment
A <- rbinom(n = n, size = 1, prob = g1W)
# E[Y | A, W] is linear in A, W
QAW <- W1 - W2 + A
# generate outcome by adding random error with N(0,1) distribution
Y <- QAW + rnorm(n = n)
```

Cross-validation

Our first exercise involves comparing two estimates of $\bar{Q}(A, W) = E[Y \mid A, W]$.

- $\bar{Q}_1$ = linear regression with main-effect terms only.
- $\bar{Q}_2$ = linear regression with all two-way interactions, quadratic terms for W_1, W_2.
- Mean squared-error risk, $R(\bar{Q}_j) = E[\{Y - \bar{Q}_j(A, W)\}^2]$.

Below, we naïvely use the full data to fit both regressions and compute risk.

```
# fit Q1 on full data
Q1_fit <- glm(Y ~ W1 + W2 + A)
# get predicted value for each observation
Q1n <- as.numeric(predict(Q1_fit))
# compute mean squared-error
risk_Q1 <- mean((Y - Q1n)^2)
# fit Q2 on full data
Q2_fit <- glm(Y ~ I(W1^2) + I(W2^2) + W1*W2 + W1*A + W2*A)
# get predicted value for each observation
Q2n <- as.numeric(predict(Q2_fit))
# compute mean squared-error
risk_Q2 <- mean((Y - Q2n)^2)
```

The estimated risk for the two estimates: $R(\bar{Q}_{n,1}) = 1.15$, $R(\bar{Q}_{n,2}) = 1.09$. We would erroneously conclude that $\bar{Q}_{n,2}$ is the better estimate!

Cross-validation

Now, we will evaluate the estimators properly using two-fold cross-validation.

```r
# first 100 observations go in split 2, second 100 in split 2
split <- c(rep(1, n/2), rep(2, n/2))
# make a data.frame of all the data
full_data <- data.frame(W1 = W1, W2 = W2, A = A, Y = Y)
# data.frame of only split 1 observations
split1 <- subset(full_data, split == 1)
# data.frame of only split 2 observations
split2 <- subset(full_data, split == 2)
# fit Q1 in split 1
Q1_fit_split1 <- glm(Y ~ W1 + W2 + A, data = split1)
# predict from split 1 fit in split 2
Q1n_split2 <- predict(Q1_fit_split1, newdata = split2)
# estimate of MSE based on split 2
risk_Q1_split2 <- mean((Y[split == 2] - Q1n_split2)^2)
# fit Q1 in split 2
Q1_fit_split2 <- glm(Y ~ W1 + W2 + A, data = split2)
# predict from split 2 fit in split 1
Q1n_split1 <- predict(Q1_fit_split1, newdata = split1)
# estimate of MSE based on split 1
risk_Q1_split1 <- mean((Y[split == 1] - Q1n_split1)^2)
# average the two estimates
cv_risk_Q1 <- (risk_Q1_split1 + risk_Q1_split2) / 2
```

Exercise

Evaluate the two-fold cross-validated risk of $\bar{Q}_2$.

```
# fit Q2 in split 1

# predict from split 1 fit in split 2

# estimate of MSE based on split 2

# fit Q2 in split 2

# predict from split 2 fit in split 1

# estimate of MSE based on split 1

# average the two estimates
```

Based on the estimated cross-validated risk, which estimator would you select?

Solution

The two-fold cross-validated risk of $\bar{Q}_2$ is computed below

```
# fit Q2 in split 1
Q2_fit_split1 <- glm(Y ~ I(W1^2) + I(W2^2) + W1*W2 + W1*A + W2*A, data = split1)
# predict from split 1 fit in split 2
Q2n_split2 <- predict(Q2_fit_split1, newdata = split2)
# estimate of MSE based on split 2
risk_Q2_split2 <- mean((Y[split == 2] - Q2n_split2)^2)
# fit Q2 in split 2
Q2_fit_split2 <- glm(Y ~ I(W1^2) + I(W2^2) + W1*W2 + W1*A + W2*A, data = split2)
# predict from split 2 fit in split 1
Q2n_split1 <- predict(Q2_fit_split2, newdata = split1)
# estimate of MSE based on split 1
risk_Q2_split1 <- mean((Y[split == 1] - Q2n_split1)^2)
# average the two estimates
cv_risk_Q2 <- (risk_Q2_split1 + risk_Q2_split2) / 2
```

Based on the estimated cross-validated risk, we would select $\bar{Q}_1$, because its cross-validated risk estimate is 1.15 compared to 1.74 for $\bar{Q}_2$.

Key takeaways

- Cross-validation is a general tool for evaluating regression estimators.
- If you can fit a regression technique, and know how to predict from the fitted regression, you can perform cross-validation.
- Fit the regression on a portion of the data; predict values on an independent portion; evaluate risk.

Ensembles of regression estimators

As we discussed in the last chapter, there is often a benefit to using weighted combinations of estimators. We now demonstrate how the weights for these estimators can be selected to minimize cross-validated risk.

```r
# a function to generate a prediction from an ensemble of Q1 and Q2
# Q1_fit and Q2_fit are fitted glm's
# newdata is a data.frame that you want to get predictions on
# Q1_weight is the weight given to the prediction from Q1_fit
# Q2_weight is 1 - Q1_weight
ensemble_predict <- function(Q1_fit, Q2_fit, newdata,
                             Q1_weight, Q2_weight = 1 - Q1_weight){
  # make sure weights approximately sum to 1
  stopifnot(abs(1 - Q1_weight - Q2_weight) < 1e-5)
  # prediction from Q1_fit on newdata
  Q1n <- predict(Q1_fit, newdata = newdata)
  # prediction from Q2_fit on newdata
  Q2n <- predict(Q2_fit, newdata = newdata)
  # weighted combination
  ensemble_prediction <- Q1_weight * Q1n + Q2_weight * Q2n
  return(as.numeric(ensemble_prediction))
}
```

Ensembles of regression estimators

A few calls to ensemble_predict to see what it does.

```
# just get data for the first observation
obs1 <- full_data[1,]
obs1
```

```
##            W1        W2 A         Y
## 1 -0.2391731 0.5980112 0 0.1411383
```

```
# get ensemble prediction for first observation using equal weights
ensemble_predict(Q1_fit = Q1_fit, Q2_fit = Q2_fit,
                 newdata = obs1, Q1_weight = 0.5)
```

```
## [1] -0.4638513
```

```
# should be the same as
Q1n[1] * 0.5 + Q2n[1] * 0.5
```

```
## [1] -0.4638513
```

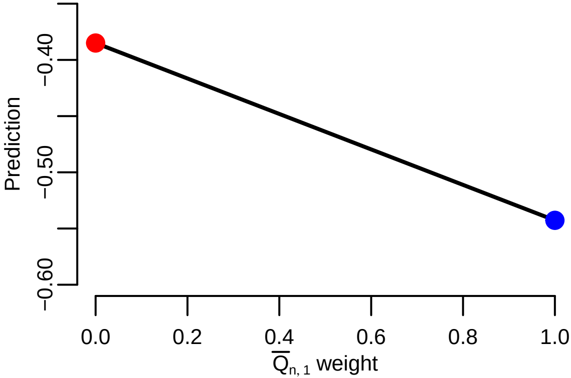
Ensembles of regression estimators

We can obtain a prediction for many possible weighting schemes.

```r
# get ensemble prediction for a sequence of weights between 0,1
weight_sequence <- seq(from = 0, to = 1, length = 500)
obs1_predict <- sapply(weight_sequence, ensemble_predict, Q1_fit = Q1_fit,
                       Q2_fit = Q2_fit, newdata = obs1)
# set graphical parameters
par(mar = c(4.1, 4.1, 0.5, 0.5), mgp = c(1.3, 0.45, 0))
plot(obs1_predict ~ weight_sequence, type = "l", lwd = 2, bty = "n",
     xlab = expression(bar(Q)[list("n",1)]*" weight"),
     ylab = "Prediction", cex.lab = 0.65, cex.axis = 0.65,
     ylim = c(-0.6, -0.35))
points(y = obs1_predict[1], x = 0, pch = 19, col = 'red', cex = 1.2)
points(y = obs1_predict[500], x = 1, pch = 19, col = 'blue', cex = 1.2)
```

Ensembles of regression estimators

We obtain a range of predictions, from only $\bar{Q}_{n,2}$ to only $\bar{Q}_{n,1}$.



Ensembles of regression estimators

We can use `ensemble_predict` to estimate cross-validated risk of each possible ensemble.

- Below, we estimate the cross-validated risk of the equally weighted ensemble.

```
# get ensemble predictions on split 2 from fits in split 1
enspred_split2 <- ensemble_predict(Q1_fit = Q1_fit_split1,
                                   Q2_fit = Q2_fit_split1,
                                   Q1_weight = 0.5,
                                   newdata = split2)
# estimate of MSE based on split 2
risk_ens_split2 <- mean((Y[split == 2] - enspred_split2)^2)
# get ensemble predictions on split 2 from fits in split 1
enspred_split1 <- ensemble_predict(Q1_fit = Q1_fit_split2,
                                   Q2_fit = Q2_fit_split2,
                                   Q1_weight = 0.5,
                                   newdata = split1)
# estimate of MSE based on split 1
risk_ens_split1 <- mean((Y[split == 1] - enspred_split1)^2)
# average the two estimates
cv_risk_ens <- (risk_ens_split1 + risk_ens_split2) / 2
```

Ensembles of regression estimators

Now we compute the cross-validated risk for many choices of weights.

```r
# define an empty vector of cv risk values
cv_risks <- rep(NA, length(weight_sequence))
# for each value of weights compute the cv risk
for(i in seq_along(weight_sequence)){
    # get ensemble predictions on split 2 from fits in split 1
    enspred_split2 <- ensemble_predict(Q1_fit = Q1_fit_split1,
                                       Q2_fit = Q2_fit_split1,
                                       Q1_weight = weight_sequence[i],
                                       newdata = split2)
    # estimate of MSE based on split 2
    risk_ens_split2 <- mean((Y[split == 2] - enspred_split2)^2)
    # get ensemble predictions on split 2 from fits in split 1
    enspred_split1 <- ensemble_predict(Q1_fit = Q1_fit_split2,
                                       Q2_fit = Q2_fit_split2,
                                       Q1_weight = weight_sequence[i],
                                       newdata = split1)
    # estimate of MSE based on split 1
    risk_ens_split1 <- mean((Y[split == 1] - enspred_split1)^2)
    # save cv risk in cv_risks vector
    cv_risks[i] <- (risk_ens_split1 + risk_ens_split2) / 2
}
```

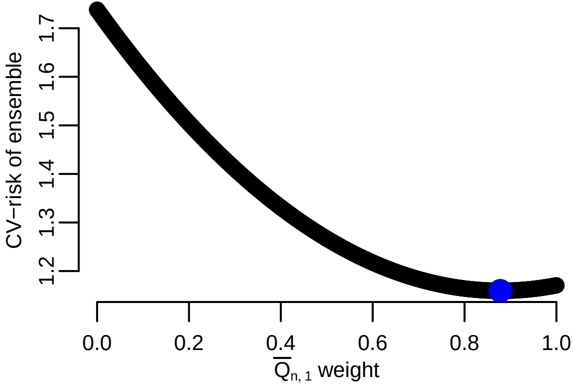
Ensemble of regression estimators

The super learner is the weighted combination with smallest cross-validated risk.

```r
# set some graphical parameters
par(mar = c(4.1, 4.1, 0.5, 0.5), mgp = c(1.3, 0.45, 0))
# plot all risks
plot(cv_risks ~ weight_sequence, bty = 'n',
     xlab = expression(bar(Q)[list("n",1)]*" weight"),
     ylab = "CV-risk of ensemble", cex.lab = 0.65, cex.axis = 0.65)
# add solid point where risk is lowest
min_idx <- which.min(cv_risks)
points(y = cv_risks[min_idx], x = weight_sequence[min_idx], col = 'blue',
       pch = 19, cex = 1.5)
```

Ensemble of regression estimators

The best cross-validated risk is attained with weight 0.88 for $\bar{Q}_1$ and 0.12 for $\bar{Q}_2$.



Key takeaways

- Super learner is a weighted combination of different regression estimates.
- We want to select weights that minimize cross-validated risk → evaluate cross-validated risk for different choices of weights.
- When > 2 estimates are considered, we generally need more clever optimization techniques to determine the best weight combination (e.g., quadratic programming).
- Once we have fit regressions in different splits, selecting weights adds little computational burden.

The SuperLearner package

The `SuperLearner` package facilitates cross-validation-based estimator selection and ensemble learning, and is available on CRAN and GitHub.

Learning objectives for today:

1. understanding and executing basic calls to `SuperLearner`;
2. writing your regression estimator that is compatible with `SuperLearner`;
3. writing your own variable screening wrapper that is compatible with `SuperLearner`;
4. evaluating the performance of super learner.

Basic calls to SuperLearner

A rundown of the most important options for the `SuperLearner` function:

- `Y` = outcome (i.e., Y for outcome regression, A for propensity score);
- `X` = covariates (i.e., (A, W) for the outcome regression, W for propensity score);
- `SL.library` = candidate regression algorithms;
- `family` = `gaussian()` or `binomial()`, changes behavior of algorithms in `SL.library`;
- `method` = specifies risk criteria and type of ensemble, good default is `"method.CC_LS` for continuous and `method.CC_nloglik` for binary outcomes;[1]
- `id` = observation identifier, cross-validation is done by `id`, so different `id`'s should imply independent observations;
- `cvControl` = controls the cross-validation procedure, most important is `V` which controls the number of folds, see `?SuperLearner.CV.control` for all controls.
- `verbose` = boolean indicating whether or not to print out messages.

[1] These are not the default values, but in my opinion are more theoretically justified than the defaults.

Basic calls to SuperLearner

Let's start by making a simple call to SuperLearner and parsing the output.

```
# set a seed for reproducibility
set.seed(123)
# basic call to SuperLearner to estimate E[Y | A, W]
sl_fit1 <- SuperLearner(Y = Y,
                        X = data.frame(W1 = W1, W2 = W2, A = A),
                        SL.library = c("SL.glm", "SL.mean"),
                        family = gaussian(),
                        method = "method.CC_LS",
                        cvControl = list(V = 2),
                        verbose = FALSE)
# see what the printed output contains
sl_fit1
```

Basic calls to SuperLearner

Let's start by making a simple call to SuperLearner and parsing the output.

```
##
## Call:
## SuperLearner(Y = Y, X = data.frame(W1 = W1, W2 = W2, A = A),
##     family = gaussian(), SL.library = c("SL.glm", "SL.mean"),
##     method = "method.CC_LS", verbose = FALSE, cvControl = list(V = 2))
##
##
##
##                   Risk       Coef
## SL.glm_All  1.253962 0.96862931
## SL.mean_All 3.344186 0.03137069
```

- Call is the call that was made to SuperLearner.
- Risk is cross-validated risk of each estimator (in this case, mean squared-error).
- Coef is the weight given to each estimator.

Basic calls to SuperLearner

We can use the code below to see what else is contained in the output.

```
# what objects are in the output
names(sl_fit1)
```

```
##  [1] "call"              "libraryNames"
##  [3] "SL.library"        "SL.predict"
##  [5] "coef"              "library.predict"
##  [7] "Z"                 "cvRisk"
##  [9] "family"            "fitLibrary"
## [11] "cvFitLibrary"      "varNames"
## [13] "validRows"         "method"
## [15] "whichScreen"       "control"
## [17] "cvControl"         "errorsInCVLibrary"
## [19] "errorsInLibrary"   "metaOptimizer"
## [21] "env"               "times"
```

Basic calls to SuperLearner

The most important output for our purposes is:

- SL.predict = vector of super learner predictions for all data;
- library.predict = matrix of predictions made by each estimator;
- coef = the weights given to each algorithm (i.e., what is printed in Coef);
- cvRisk = the cross-validated risk of each algorithm (i.e., what is printed in Risk);
- fitLibrary = the fitted regression estimators (e.g., glm objects);
- validRows = a list of which id's were in the validation fold.

Basic calls to SuperLearner

We can also obtain super learner predictions on new data.

```
# a new observation
new_obs <- data.frame(A = 1, W1 = 0, W2 = 0)
# call predict using sl_fit1
new_predictions <- predict(sl_fit1, newdata = new_obs)
# super learner prediction is in $pred
new_predictions$pred
```

```
##           [,1]
## [1,] 0.8570446
```

```
# the prediction made by each estimator is in $library.predict
new_predictions$library.predict
```

```
##      SL.glm_All SL.mean_All
## [1,]  0.8883005  -0.1080424
```

Writing your own regression

Let's take a closer look at how regression estimators are passed to the `SL.library` option.

- We set `SL.library = c("SL.glm","SL.mean")`, which generated an ensemble of a main terms linear regression (`SL.glm`) and an intercept-only regression (`SL.mean`).
- `SL.glm` and `SL.mean` are functions that are included in the `SuperLearner` package.
- Many methods are included and can be viewed using `listWrappers(what = "SL")`.

Even though the list of potential algorithms is fairly extensive, we may also wish to specify our own regressions.

- E.g., want to fit a specific formula for a regression.
- E.g., want to try different tuning parameters for a machine learning algorithm.

We need to write our own functions that play nicely with `SuperLearner`!

Writing your own regression

As an example, let's take a look at the SL.glm function to see it's format.

```
SL.glm
```

```
## function (Y, X, newX, family, obsWeights, model = TRUE, ...)
## {
##     if (is.matrix(X)) {
##         X = as.data.frame(X)
##     }
##     fit.glm <- glm(Y ~ ., data = X, family = family, weights = obsWeights,
##         model = model)
##     if (is.matrix(newX)) {
##         newX = as.data.frame(newX)
##     }
##     pred <- predict(fit.glm, newdata = newX, type = "response")
##     fit <- list(object = fit.glm)
##     class(fit) <- "SL.glm"
##     out <- list(pred = pred, fit = fit)
##     return(out)
## }
## <bytecode: 0x7f975c673060>
## <environment: namespace:SuperLearner>
```

Writing your own regression

The key inputs that are taken by a `SuperLearner` wrapper function are

- `Y` = vector of outcomes used to fit the regression;
- `X` = matrix of covariates used to fit the regression;
- `newX` = matrix of covariates that we use to get predictions from the fitted regression;
- `family` = `gaussian()` or `binomial()` → linear or logistic regression;
- `...` = allows other arguments to be passed in without breaking.

The general work flow of a `SuperLearner` wrapper function is

- use `Y` and `X` to fit a regression
    - when called from `SuperLearner` these will contain the training data for a given split;
- use the fitted regression to get predictions on `newX`;
- format the output as a list with named entries `pred` and `fit`:
    - `pred` = predictions on `newX`
    - `fit` = anything that is needed to get predictions at a later time or that we want to study in the `$fitLibrary` output later.

Writing your own regression

The same general work flow can be found in any of wrapper functions.

```
SL.mean
```

```
## function (Y, X, newX, family, obsWeights, id, ...)
## {
##     meanY <- weighted.mean(Y, w = obsWeights)
##     pred <- rep.int(meanY, times = nrow(newX))
##     fit <- list(object = meanY)
##     out <- list(pred = pred, fit = fit)
##     class(out$fit) <- c("SL.mean")
##     return(out)
## }
## <bytecode: 0x7f97602c6720>
## <environment: namespace:SuperLearner>
```

Writing your own regression

We can use the same work flow to define our own regression estimators.

- Below, we define a new function that includes quadratic forms for W_1 and W_2.
- The class of the object and the silly message will be discussed presently.

```
SL.quadglm <- function(Y, X, newX, family, ...){
    # this line assumes X will have a named columns W1, W2, and A
    # we are also ignoring the family input, assuming that we will be
    # using this function for linear regression only
    quadglm_fit <- glm(Y ~ W1 + I(W1^2) + W2 + I(W2^2) + A, data = X)
    # get predictions on newX
    pred <- predict(quadglm_fit, newdata = newX)
    # format the output as named list
    fit <- list(fitted_model = quadglm_fit, message = "Hello world!")
    out <- list(fit = fit, pred = pred)
    # give the object a class
    class(out$fit) <- "SL.quadglm"
    # return the output
    return(out)
}
```

Writing your own regression

We can now include this wrapper in our SL.library.

```
# set a seed for reproducibility
set.seed(123)
# call to SuperLearner that includes our own wrapper
sl_fit2 <- SuperLearner(Y = Y,
                        X = data.frame(W1 = W1, W2 = W2, A = A),
                        SL.library = c("SL.glm", "SL.mean", "SL.quadglm"),
                        family = gaussian(),
                        method = "method.CC_LS",
                        cvControl = list(V = 2),
                        verbose = FALSE)
```

Writing your own regression

Recall, we said that the fit object of the wrapper should include

- anything that we want to study in the $fitLibrary output later, and
- anything that is needed to get predictions at a later time.

Let's find our message in the output of SuperLearner.

```
sl_fit2$fitLibrary$SL.quadglm$message
```

```
## [1] "Hello world!"
```

What about predicting from a SuperLearner object that contains a custom wrapper?

```
predict(sl_fit2, newdata = new_obs)
```

```
## Error in UseMethod("predict") :
no applicable method for 'predict' applied to an object of class "SL.quadglm"
```

Writing your own regression

We need to tell `R` how to predict from our regression.

- Specifically, we need a `predict` method for objects of class `"SL.quadglm"`.

In `R`, when we call `predict(some_object)` where `some_object` has class `some_class`, `R` will attempt to execute `predict.some_class(some_object)`.

- The previous error is because we have not defined a function `predict.SL.quadglm`.
- Such functions (referred to as `method`'s) need to have a specific format.
- In this case, it needs to *at least* take as input `object` and `...` .

```r
predict.SL.quadglm <- function(object, newdata, ...){
    # object will be the $fit entry of the output from SL.quadglm
    # since class(object$fitted_model) = "glm", this will in turn call
    # predict.glm to obtain predictions
    pred <- predict(object$fitted_model, newdata = newdata)
    # return predictions
    return(pred)
}
```

Writing your own regression

We can now make new predictions from the SuperLearner object with our custom wrapper.

```
# this calls predict.SuperLearner(object = sl_fit2, ...)
sl2_pred <- predict(sl_fit2, newdata = new_obs)
# super learner prediction
sl2_pred$pred
```

```
##           [,1]
## [1,] 0.8570446
```

```
# library predictions
sl2_pred$library.predict
```

```
##      SL.glm_All SL.mean_All SL.quadglm_All
## [1,]  0.8883005  -0.1080424      0.7007979
```

Exercise

Write a wrapper and predict method that fits the correct propensity score.

```
# define wrapper function
SL.truepropensity <- function(Y, X, newX, family, ...){
    # fit logistic regression model with interaction between W1 & W2
    # get predictions on newX
    # format output
    # return output
}

# define predict method
predict.SL.truepropensity <- function(object, newdata, ...){
    # predict from object
    # return predictions
}
```

Exercise

Evaluate this fit on its negative log-likelihood versus a main terms logistic regression and an intercept-only logistic regression.
Generate the corresponding super learner based on a logistic-linear ensemble.

```
sl_fit3 <- SuperLearner(Y = ,
                        X = ,
                        family = ,
                        SL.library = ,
                        # use negative log-likelihood loss
                        method = "method.CC_nloglik",
                        cvControl = list(V = 2),
                        verbose = FALSE)
```

Solution

Write a wrapper and predict method that fits the correct propensity score.

```r
# define wrapper function
SL.truepropensity <- function(Y, X, newX, family, ...){
    # fit logistic regression model with interaction between W1 & W2
    fitted_model <- glm(Y ~ W1*W2, data = X, family = binomial())
    # get predictions on newX
    # type = 'response' is key to getting predictions on the correct scale!
    pred <- predict(fitted_model, newdata = newX, type = "response")
    # format output
    out <- list(fit = list(fitted_model = fitted_model), pred = pred)
    # return output
    return(out)
}

# define predict method
predict.SL.truepropensity <- function(object, newdata, ...){
    # predict from object
    pred <- predict(object$fitted_model, newdata = newdata, type = "response")
    # return predictions
    return(pred)
}
```

Solution

Evaluate this fit on its negative log-likelihood versus a main terms logistic regression and an intercept-only logistic regression.
Generate the corresponding super learner based on a logistic-linear ensemble.

```
# set seed
set.seed(1234)
# fit super learner
sl_fit3 <- SuperLearner(Y = A,
                        X = data.frame(W1 = W1, W2 = W2),
                        family = binomial(),
                        SL.library = c("SL.glm","SL.mean","SL.truepropensity"),
                        # use negative log-likelihood loss
                        method = "method.CC_nloglik",
                        cvControl = list(V = 2),
                        verbose = FALSE)
```

Solution

```
sl_fit3
```

```
##
## Call:
## SuperLearner(Y = A, X = data.frame(W1 = W1, W2 = W2), family = binomial(),
##     SL.library = c("SL.glm", "SL.mean", "SL.truepropensity"),
##     method = "method.CC_nloglik", verbose = FALSE, cvControl = list(V = 2))
##
##
##
##                             Risk       Coef
## SL.glm_All             78.73961 0.31304504
## SL.mean_All           115.23620 0.06860934
## SL.truepropensity_All  77.90696 0.61834562
```

Comments on exercise

A few comments on the exercise:

- When `method = "method.CC_nloglik"`, we are fitting the logistic-linear ensemble

$$g_{n,\omega} = \text{expit}\left[\sum_{k=1}^{K} \omega_k \text{logit}(g_{n,k}) \right] .$$

- The scale of negative log-likelihood and mean squared-error are likely to be different.
- Super learner may give weight to algorithms with poor `Risk`. This does not necessarily mean it is over-fitting! The cross-validated risks are the best description of how well an algorithm performs; not how much weight it receives in the super learner.
- What is a "good" `Risk`? The lowest one we can possibly get!
- For mean squared-error, 1 - `Risk/var(Y)` gives an R^2-ish metric.

Variable screening

You may have noticed that SuperLearner output adds _All to the output for each algorithm (e.g., SL.glm_All).

- By default, SuperLearner assumes you want All variables in X to be included.

Often, we wish to do variable screening as part of the learning process. For example, we may wish to consider a library consisting of

- estimator 1 = linear regression with all variables;
- estimator 2 = linear regression with a-priori selected group of variables;
- estimator 3 = linear regression with only variables that are significantly correlated with the outcome.

We can combine variable screening procedures with different regression techniques to vastly increase the number of candidate regressions!

- SuperLearner includes easy syntax for doing so.

Variable screening

SuperLearner includes several screening procedures. These can be viewed via listWrappers(what = "screen").

The key inputs that are taken by a SuperLearner variable screening function are

- Y = vector of outcomes used to fit the regression;
- X = matrix of covariates used to fit the regression;
- family = gaussian() or binomial();
- ... = allows other arguments to be passed in without breaking.

The general work flow of a SuperLearner variable screening function is

- use Y and X to do some variable screening procedure
- format output as vector of TRUE/FALSE indicating what columns of X to include.

Variable screening

Let's see what `screen.corP` does.

```
screen.corP
```

```
## function (Y, X, family, obsWeights, id, method = "pearson", minPvalue = 0.1,
##     minscreen = 2, ...)
## {
##     listp <- apply(X, 2, function(x, Y, method) {
##         ifelse(var(x) <= 0, 1, cor.test(x, y = Y, method = method)$p.value)
##     }, Y = Y, method = method)
##     whichVariable <- (listp <= minPvalue)
##     if (sum(whichVariable) < minscreen) {
##         warning("number of variables with p value less than
##         whichVariable[rank(listp) <= minscreen] <- TRUE
##     }
##     return(whichVariable)
## }
## <bytecode: 0x7f9760c0b8c8>
## <environment: namespace:SuperLearner>
```

Variable screening

To include screening wrappers in a call to `SuperLearner` we change the way we input `SL.library`.

- Previously, we used a vector of regression functions.
- Now, we use a `list` of form `c(regression, screen)`.

```
sl_fit4 <- SuperLearner(Y = Y,
                        X = data.frame(W1 = W1, W2 = W2, A = A),
                        SL.library = list(c("SL.glm", "All"),
                                          c("SL.glm", "screen.corP"),
                                          c("SL.mean", "All")),
                        family = gaussian(),
                        method = "method.CC_LS",
                        cvControl = list(V = 2),
                        verbose = FALSE)
```

Variable screening

The output now shows two `SL.glm` regression, one that used `All` variables and another that used only those significantly correlated with the outcome.

```
sl_fit4
```

```
##
## Call:
## SuperLearner(Y = Y, X = data.frame(W1 = W1, W2 = W2, A = A),
##     family = gaussian(), SL.library = list(c("SL.glm",
##         "All"), c("SL.glm", "screen.corP"), c("SL.mean",
##         "All")), method = "method.CC_LS", verbose = FALSE,
##     cvControl = list(V = 2))
##
##
##                           Risk      Coef
## SL.glm_All          1.449263 0.2339346
## SL.glm_screen.corP 1.388447 0.7660654
## SL.mean_All         3.734942 0.0000000
```

Variable screening

Only include covariates that change treatment coefficient by more than 10%.

```r
screen_confounders <- function(Y, X, family, trt_name = "A",
                               change = 0.1, ...){
    # fit treatment only model & get coefficient for treatment
    fit_init <- glm(as.formula(paste0("Y ~ ", trt_name)),
                    data = X, family = family)
    trt_coef <- fit_init$coef[2]
    # identify which column of X is the trt variable
    trt_col <- which(colnames(X) == trt_name)
    # set all variables except trt to not be included initially
    include <- rep(FALSE, ncol(X)); include[trt_col] <- TRUE
    # loop over variables in X other than trt
    for(j in seq_len(ncol(X))[-trt_col]){
        # find variable name
        var_name <- colnames(X)[j]
        # fit trt + variable model, get trt coefficient
        fit <- glm(as.formula(paste0("Y ~ ", trt_name, "+", var_name)),
                   data = X, family = family)
        new_trt_coef <- fit$coef[2]
        # check if changed more than specified amount
        include[j] <- abs((new_trt_coef - trt_coef)/trt_coef) > change
    }
    return(include)
}
```

Variable screening

```
sl_fit5 <- SuperLearner(Y = Y,
                        X = data.frame(W1 = W1, W2 = W2, A = A),
                        SL.library = list(c("SL.glm", "screen_confounders"),
                                          c("SL.mean", "All")),
                        family = gaussian(),
                        method = "method.CC_LS",
                        cvControl = list(V = 2),
                        verbose = FALSE)
# print results
sl_fit5
```

```
##
## Call:
## SuperLearner(Y = Y, X = data.frame(W1 = W1, W2 = W2, A = A),
##     family = gaussian(), SL.library = list(c("SL.glm",
##         "screen_confounders"), c("SL.mean", "All")), method = "method.CC_LS",
##     verbose = FALSE, cvControl = list(V = 2))
##
##
##                               Risk        Coef
## SL.glm_screen_confounders 1.215335 0.998181331
## SL.mean_All               2.955453 0.001818669
```

Evaluating the super learner

Often, we would like to objectively evaluate the performance of the super learner.

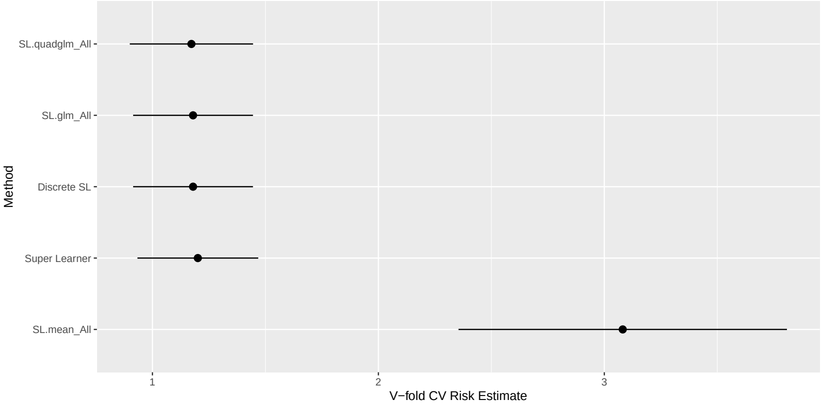
- Theory says we do well eventually, but did we do well in these data?
- The risk of the super learner can be estimated using cross-validation.
- In the SuperLearner package, this is achieved via the CV.SuperLearner function.

```
# set seed for reproducibility
set.seed(1234)

# two-fold cross-validated evaluation of two-fold super learner
cv_sl <- CV.SuperLearner(
  Y = Y, X = data.frame(W1 = W1, W2 = W2, A = A),
  family = gaussian(),
  method="method.CC_LS",
  SL.library = c("SL.glm","SL.mean","SL.quadglm"),
  cvControl = list(V = 2),
  innerCvControl = list(list(V = 2),
                        list(V = 2))
)
# plot the results
plot(cv_sl)
```

Evaluating the super learner

The plot displays the cross-validated risk estimate and 95% confidence interval.



General recommendations

Analysis

- Generally, `V = 5` or `V = 10` folds is adequate.
- Include your own regressions.
- Include flexible learners – `SL.earth` (polynomial multivariate adaptive regression splines), `SL.gbm` (gradient boosted machines), and `SL.ranger` (fast random forests) are good examples.
- Consider permuting treatment assignment when developing code to remain objective.

Flexibility vs. computer time

- Do not expect things to run < 1 minute; a thorough analysis will take time.
- Time how long each wrapper takes to run ahead of time; `system.time` and option `verbose = TRUE` are your friends. Consider eliminating estimators that take too long.
- `SL.step` and `SL.step.interaction` can take much longer than expected.

Reporting results

- Include number of folds.
- Describe what algorithms were included.
- Describe how ensemble was constructed/how estimator selected.
- Cross-validated risk of super learner relative to candidate estimators (sensitivity).

General recommendations

Example write-up of super learner analysis

Methods

The outcome regression was estimated using super learning. This approach estimates the linear combination of regression estimators that minimizes ten-fold cross-validated mean squared-error. We included twenty five regression estimators in the super learner (Table 1, Appendix A) that included, among others, linear regression models, stepwise regression models, random forests, and polynomial multivariate regression splines.

The propensity score was also fit using super learning. However, in this case we estimated the logistic-linear combination of regression estimators that minimized ten-fold cross-validated negative log-likelihood loss. We included twenty regression estimators (Table 2, Appendix A) that included, among others, logistic regression models and flexible machine learning techniques.

In sensitivity analyses, we estimated the ten-fold cross-validated risk of the two super learners relative to the other candidate regression estimators.

General recommendations

Example write-up of super learner analysis

Results

The super learner for the outcome regression gave meaningful weight to five estimators (Table 1, Appendix A), with the greatest weight given to the stepwise regression model. The super learner for the propensity score gave meaningful weight to three these estimators (Table 2, Appendix A), with greatest weight given to the random forest regression.

We found that the super learner for the outcome regression had better cross-validated risk than any single regression we considered (Figure 1, Appendix A). For the propensity score, we found that the random forest regression had the best cross-validated risk, though the performance of super learner was comparable (Figure 2, Appendix A).

General recommendations

Example write-up of super learner analysis

Appendix

Function name	Description	SL weight
`SL.glm_All`	main-terms linear regression using all variables	0.00
`SL.glm_screen.corP`	main-terms linear regression using only variables with significant correlation with outcome (p-value < 0.05)	0.12
`SL.ranger_All`	random forest with "default" tuning parameters	0.01
`SL.mean_All`	intercept-only regression	0.00
...	...	...
`SL.gbm_All`	gradient boosted regression trees with "default" tuning parameters	0.45