Quantitative Research Methods IV - 17.806

Recitation, Week 5.

Topic: Supervised Learning III.

Benjamín Muñoz

March 10, 2023

MIT

Massachusetts Institute of Technology

Table of contents

1. Introduction

2. Review of Concepts

3. Problem Set 2

1/ Introduction

Introduction

- Problem set 2 due: March 20, 3:30pm.
- Office hour: Every Tuesday 3:30-5pm.
- Piazza for asking questions: piazza.com
- Use of Markdown/LaTeX is recommended but not required.
    - Introduction.
    - Extra Resources.
    - Reference Guide.
- Pro-Tip: Take advantage of chunk options:
    - `eval = TRUE/FALSE` .
    - `echo = TRUE/FALSE`.

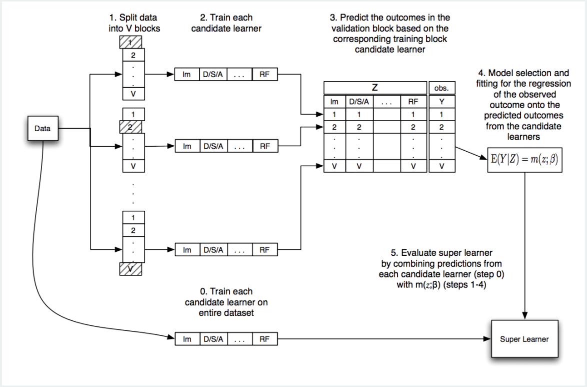
Problem Set General Reminders

- Collaborators: You must note all of your collaborators. If you work alone, please indicate that as well.
- Annotating code: Please annotate your code thoroughly so that other people including me can easily follow each step. This is an important practice especially when you do collaborative research.
- Embedding code: If you use Rmarkdown or Rsweave, please make sure that lines do not get cut off. You can avoid this by `tidy`=TRUE option in code chunks.

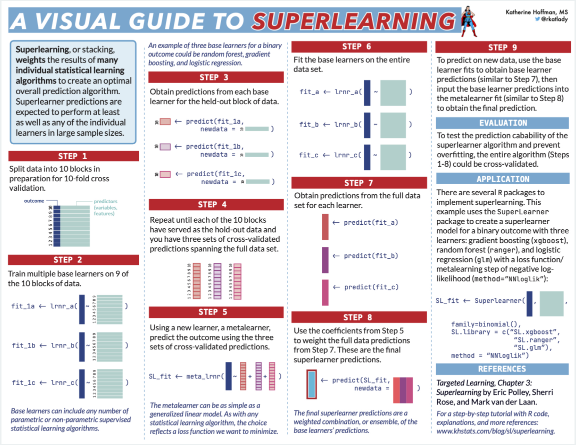
2/ Review of Concepts

Review of Concepts

- Super-Learning Workflow.
- R Package: `SuperLearner`.
- Reference Guide.


1. Split data into V blocks
2. Train each candidate learner
3. Predict the outcomes in the validation block based on the corresponding training block candidate learner
4. Model selection and fitting for the regression of the observed outcome onto the predicted outcomes from the candidate learners
5. Evaluate super learner by combining predictions from each candidate learner (step 0) with m(z;β) (steps 1-4)
0. Train each candidate learner on entire dataset
Data
lm
D/S/A
...
RF
Z
obs.
Y
1
2
V
$E(Y|Z) = m(z;\beta)$
Super Learner

Review of Concepts



A VISUAL GUIDE TO SUPERLEARNING

Katherine Hoffman, MS
@rkatlady

Superlearning, or stacking, **weights** the results of **many individual statistical learning algorithms** to create an optimal overall prediction algorithm. Superlearner predictions are expected to perform at least as well as any of the individual learners in large sample sizes.

STEP 1

Split data into 10 blocks in preparation for 10-fold cross validation.

STEP 2

Train multiple base learners on 9 of the 10 blocks of data.

```
fit_1a ← lrnr_a(  ~  )
fit_1b ← lrnr_b(  ~  )
fit_1c ← lrnr_c(  ~  )
```

Base learners can include any number of parametric or non-parametric supervised statistical learning algorithms.

An example of three base learners for a binary outcome could be random forest, gradient boosting, and logistic regression.

STEP 3

Obtain predictions from each base learner for the held-out block of data.

```
10 ← predict(fit_1a,
       newdata = 10  )
10 ← predict(fit_1b,
       newdata = 10  )
10 ← predict(fit_1c,
       newdata = 10  )
```

STEP 4

Repeat until each of the 10 blocks have served as the hold-out data and you have three sets of cross-validated predictions spanning the full data set.

STEP 5

Using a new learner, a metalearner, predict the outcome using the three sets of cross-validated predictions.

```
SL_fit ← meta_lrnr(  ~  +  +  )
```

The metalearner can be as simple as a generalized linear model. As with any statistical learning algorithm, the choice reflects a loss function we want to minimize.

STEP 6

Fit the base learners on the entire data set.

```
fit_a ← lrnr_a(  ~  )
fit_b ← lrnr_b(  ~  )
fit_c ← lrnr_c(  ~  )
```

STEP 7

Obtain predictions from the full data set for each learner.

```
← predict(fit_a)
← predict(fit_b)
← predict(fit_c)
```

STEP 8

Use the coefficients from Step 5 to weight the full data predictions from Step 7. These are the final superlearner predictions.

```
← predict(SL_fit,
     newdata =  )
```

The final superlearner predictions are a weighted combination, or ensemble, of the base learners' predictions.

STEP 9

To predict on new data, use the base learner fits to obtain base learner predictions (similar to Step 7), then input the base learner predictions into the metalearner fit (similar to Step 8) to obtain the final prediction.

EVALUATION

To test the prediction cabability of the superlearner algorithm and prevent overfitting, the entire algorithm (Steps 1-8) could be cross-validated.

APPLICATION

There are several R packages to implement superlearning. This example uses the `SuperLearner` package to create a superlearner model for a binary outcome with three learners: gradient boosting (`xgboost`), random forest (`ranger`), and logistic regression (`glm`) with a loss function/ metalearning step of negative log-likelihood (`method="NNloglik"`):

```
SL_fit ← Superlearner(  ,  ,
    family=binomial(),
    SL.library = c("SL.xgboost",
                   "SL.ranger",
                   "SL.glm"),
    method = "NNloglik")
```

REFERENCES

Targeted Learning, Chapter 3: Superlearning by Eric Polley, Sherri Rose, and Mark van der Laan.

For a step-by-step tutorial with R code, explanations, and more references: www.khstats.com/blog/sl/superlearning.

- Tutorial 1.
- Tutorial 2.

Causal Machine Learning

1. High-Dimensionality.
    - Selection of Confounders.
    - Selection of Instruments.
    - Higher order Interactions of Treatments.
2. Heterogeneous Treatment Effects.
    - Subgroup Analysis.

3/ Problem Set 2

Problem Set 2 Goals

- Goal of the first section:
    - Conduct cross validation exercise by hand.
    - Visually check overfitting problem.
    - Learn how to do simulations (e.g., create data, visualize results).
- Goal of the second section:
    - Apply various ML tools to political science questions.
    - Practice their implementation in R.
    - Compare their performance and draw substantial lessons.

Section 1: Indicator function and Polynomials

- Indicator function: $\mathbb{I}(\text{condition})$

R Code
```
x <- c(-2, -3, -1, 0, 1, 2, 3, 4, 5)
x < 0
## [1] TRUE TRUE TRUE FALSE FALSE FALSE FALSE FALSE FALSE
(x < 0) * 2
## [1] 2 2 2 0 0 0 0 0 0
```

- e.g.) 4-th degree (order) polynomial regression

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 x^4 + \epsilon$$

- Terms x and x^2 will be highly correlated where $x > 0$
- `poly` function creates a "curved" set of variables from quadratic terms and makes sure that each term is not highly correlated to each other
- Technically, `poly` with the default setting computes "orthogonal polynomials" where each polynomial is orthogonal

Section 1: Indicator function and Polynomials

R Code

```
#Load data
data(cars)

#Run Model
m1 <- lm(dist ~ speed + I(speed^2), data = cars)
m2 <- lm(dist ~ poly(speed, degree = 2, raw = FALSE), data = cars)
m3 <- lm(dist ~ poly(speed, degree = 2, raw = TRUE), data = cars)

# Correlations
cor(model.matrix(m1)[, 2], model.matrix(m1)[, 3])
## [1] 0.9794765
cor(model.matrix(m2)[, 2], model.matrix(m2)[, 3])
## [1] 4.686464e-17
cor(model.matrix(m3)[, 2], model.matrix(m3)[, 3])
## [1] 0.9794765
```

	Model 1	Model 2	Model 3
(Intercept)	2.47	42.98^{***}	2.47
	(14.82)	(2.15)	(14.82)
speed	0.91		
	(2.03)		
speed^2	0.10		
	(0.07)		
poly(speed, degree = 2, raw = FALSE)1		145.55^{***}	
		(15.18)	
poly(speed, degree = 2, raw = FALSE)2		23.00	
		(15.18)	
poly(speed, degree = 2, raw = TRUE)1			0.91
			(2.03)
poly(speed, degree = 2, raw = TRUE)2			0.10
			(0.07)

$^{***}p < 0.001$; $^{**}p < 0.01$; $^{*}p < 0.05$

Section 1: Cross-Validation

- How to split data?
- Approach 1: randomly assign folds to each observation.
- This does not ensure that the size of each fold is equal.
- Practically ok if your N is large enough compared to K.
- Totally fine for this problem set.

R Code

```r
set.seed(17.806)
# 5-folds cross validation with 10 observations
K <- 5; N <- 10; x <- rnorm(N); y <- rnorm(N)
folds <- sample(1:K, size = N, replace = TRUE) # assign folds
x[folds == 5]; y[folds == 5] # fifth folds: four observation
## [1] -0.07963674 -0.81726793 -0.16561194 0.36658112
## [1] 0.64319207 0.18791807 -0.05517906 0.63358473
x[folds == 2]; y[folds == 2] # second folds: one observations
## [1] -0.232987
## [1] 1.295322
```

Section 1: Cross-Validation

- How to make the fold size equal?
- Approach 2: randomly assign IDs to each observation and select these observations.
- By doing so, you can ensure that the size is equal.
- In the solution, I show how to implement this efficiently.

R Code

```r
# assign folds
IDs <- sample(1:N, size = N, replace = FALSE)
# there should be 2 obs. in each fold
# ID 1:2 belongs to the first fold, 3:4 belongs to the second fold ...
x[IDs %in% c(1:2)] # first fold
## [1] 0.7720908 1.7165340
x[IDs %in% c(3:4)] # second fold
## [1] 0.9728744 0.3665811
```

Section 1: Plot estimated functions

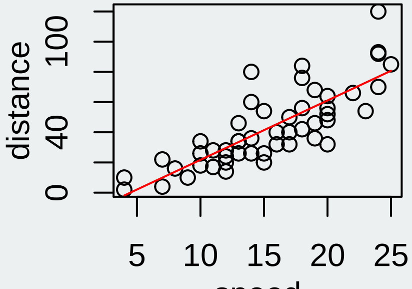
R Code

```
# Load data
data(cars)

#Run model
m1 <- lm(dist ~ speed, cars)

#Create prediction
dist_hat <- predict(m1) # generate prediction

#Create Figure
par(mar = c(4, 4, 1, 1))
plot(x = cars$speed, y = cars$dist, ann = FALSE)
lines(x = cars$speed, y = dist_hat, col = "red")
title(ylab = "distance", line = 2)
title(xlab = "speed", line = 2)
```



Section 2: ML Algorithms

- All the ML algorithms have easy-to-implement R packages/functions!

R Code

```r
 # LASSO
lasso <- glmnet(x = xtrain, y = ytrain, family = 'binomial', lambda = 0.1)
# check out the cv.glmnet also!

# random forest
rf <- randomForest(x = xtrain, y = factor(ytrain))

# neural network
nn <- nnet(x = xtrain, y = ytrain, size = 10, entropy = TRUE, trace = FALSE)

# svm linear kernel
svm_lin <- svm(x = xtrain, y = factor(ytrain), kernel = "linear")

# svm radial kernel
svm_rbf <- svm(x = xtrain, y = factor(ytrain), kernel = "radial")

#xgboost
xgb_train <- xgb.DMatrix(data = as.matrix(xtrain), label = ytrain)
xgb_test <- xgb.DMatrix(data = as.matrix(xtest), label = ytest)
xgb_model <- xgb.train(data = xgb_train, nrounds = 5000, verbose = 1)
```

Section 2: LASSO

- Use the package `glmnet`.
    - `x`: input matrix.
    - `y`: target vector.
    - `family`: type of model.
    - `alpha`: mixing parameter.
    - `lambda`: penalty parameter.
    - `standardize`: standardization of X (default = TRUE).
    - `exclude`: exclude variables from regularization/variable selection.
    - `maxit`: number of iterations.

R Code

```
#Load package
library(glmnet)

#Load data
data(QuickStartExample)
x <- QuickStartExample$x
y <- QuickStartExample$y

#Run learner
l1 <- glmnet(x=x, y=y, family = "gaussian", lambda = 0) #Ridge
l2 <- glmnet(x=x, y=y, family = "gaussian", lambda = 1) #LASSO
```

Section 2: Random Forest

- Use the package `randomForest` o `ranger`.
  - `x:` input matrix.
  - `y:` target vector.
  - `xtest:` and `ytest`.
  - `ntree:` number of trees.
  - `mtry:` number of variables randomly sampled.
  - `replace:` sampling with or without replacement.
  - `cutoff:` threshold point (only for classification).
  - `sampsize:` number of samples, `nodesize:` maximum number of terminal nodes.