Bob Bell

11.13.2013

PH252D

R Assignment 3

2 Import and explore the data set Rassign3.Fa2013.csv

2.1

ObsData <- read.csv('Rassign3.Fa2013.csv')

2.2

> names(ObsData)

[1] "W1" "W2" "W3" "W4" "W5" "Y"

> summary(ObsData)

W1 W2 W3

Min. :0.000 Min. :0.0000 Min. :0.000327

1st Qu.:0.000 1st Qu.:0.0000 1st Qu.:1.194527

Median :0.000 Median :1.0000 Median :2.486190

Mean :0.108 Mean :0.5082 Mean :2.474991

3rd Qu.:0.000 3rd Qu.:1.0000 3rd Qu.:3.731702

Max. :1.000 Max. :1.0000 Max. :4.998937

W4 W5 Y

Min. :0.1192 Min. :1.000 Min. : 88.00

1st Qu.:0.2839 1st Qu.:1.000 1st Qu.: 99.28

Median :0.3980 Median :2.000 Median :109.39

Mean :0.4174 Mean :1.897 Mean :109.39

3rd Qu.:0.5522 3rd Qu.:2.000 3rd Qu.:119.30

Max. :0.8807 Max. :4.000 Max. :137.65

> tail(ObsData)

W1 W2 W3 W4 W5 Y

4995 0 0 0.60873216 0.3926807 2 92.45187

4996 1 0 1.47052461 0.7134672 3 95.65791

4997 0 0 0.08428971 0.4476780 2 90.35491

4998 0 0 1.12818655 0.1597886 4 97.52296

4999 0 1 0.20743929 0.4439766 1 112.35356

5000 0 0 0.43452632 0.1213190 1 98.99388

2.3

> n <- nrow(ObsData)

3 Code discrete SuperLearner to select the estimator with the lowest cross-validated risk

3.1 Discrete SL allows us to choose the best candidate estimator based on the lowest cross-validated risk (minimizing according to expected squared error loss function in this case). We have no idea a priori which specific parametric model best predicts the outcome, given our treatment and covariates. We need discrete SuperLearner to choose from our library of parametric models in a supervised way; trying many different models in an ad-hoc way could lead to biased point estimates and misleading inference.

3.2

> sinW3<- sin(ObsData$W3)

> W4sq <- ObsData$W4\*ObsData$W4

> cosW5 <- cos(ObsData$W5)

> ObsData <- data.frame(ObsData, sinW3, W4sq, cosW5)

3.3 Split data into 20 folds

> Fold <- c(rep(1, n/20))

> for(V in 2:20){

+ Fold<- c(Fold, rep(V, n/20))

+ }

ObsData<- data.frame(ObsData, Fold)

> head(ObsData)

W1 W2 W3 W4 W5 Y Fold

1 0 1 1.3154797 0.3032379 1 117.38032 1

2 0 1 4.0185981 0.6308452 3 122.84623 1

3 1 1 0.1646284 0.6374620 1 117.85469 1

4 0 1 3.1764225 0.2709290 2 124.97231 1

5 0 0 2.8301433 0.3358813 3 96.78662 1

6 0 1 2.5680567 0.5695696 1 114.35169 1

3.4 Create an empty matrix CV.risk with 20 rows and 4 columns for each algorithm, evaluated

at each fold

CV.risk<- matrix(NA, nrow=20, ncol=4)

3.5

for(V in 1:20){

valid<- ObsData[Fold==V, ]train<- ObsData[Fold !=V, ]

EstA<- glm(Y~ W1+W2+sinW3+W4sq, data=train)

EstB<- glm(Y~ W1+W2+W4+cosW5, data=train)

EstC<- glm(Y~ W2\*W5+W3+W4sq+cosW5, data=train)

EstD<- glm(Y~ W1\*W2\*W5, data=train)

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')

CV.risk[V,]<- c(mean((valid$Y - PredA)^2), mean((valid$Y - PredB)^2), mean((valid$Y - PredC)^2), mean((valid$Y - PredD)^2)) }

3.6

> colMeans(CV.risk)

[1] 8.312289 13.025325 7.762348

[4] 16.031240

Estimator C is the best…

3.7 Run this on all of the data.

EstC.all<- glm(Y~ W2\*W5+W3+W4sq+cosW5, data=ObsData)

> EstC.all

Call: glm(formula = Y ~ W2 \* W5 + W3 + W4sq + cosW5, data = ObsData)

Coefficients:

(Intercept) W2 W5

95.64175 12.84349 0.07781

W3 W4sq cosW5

2.02159 -10.80041 2.11408

W2:W5

4.87470

Degrees of Freedom: 4999 Total (i.e. Null); 4993 Residual

Null Deviance: 618800

Residual Deviance: 38700 AIC: 24440

We can probably do better with SuperLearner. Can develop an algorithm that is the convex weighted combination of a number of candidate algorithms. This weighted average of regressions is possibly better than any one of the algorithms alone.

4

4.1 Load SuperLearner and set seed  
> library('SuperLearner')

Loading required package: nnls

Super Learner

Version: 2.0-10

Package created on 2013-09-03

Use SuperLearnerNews() to see changes from previous versions and latest news

Warning message:

package ‘SuperLearner’ was built under R version 2.15.3

> set.seed(252)

4.2 Wrapper Function

> source('Rassign3.Fa2013.Wrappers.R')

4.3 Specify Algorithms to be included

> SL.library<- c('SL.glm.EstA', 'SL.glm.EstB', 'SL.glm.EstC', 'SL.glm.EstD', 'SL.ridge','SL.rpartPrune', 'SL.polymars', 'SL.mean')

4.3 Bonus – explain algorithms SL.ridge, SL.rpartPrune, SL.polymars and SL.mean.

SL.ridge: We fit a linear model by ridge regression. When it is difficult to measure predictor variables perfectly, it may seem plausible to include multiple measurements for each one of them. Multiple measurements of the same variable may be (strongly) correlated leading to linear regression that is nearly multi-collinear with multiple, mutually-canceling large coefficients. This makes regression hard to interpret and its predictions hard to trust. One way to cope with this is to add a penalty for large coefficient values. This is the essence behind ridge regression or Tikhonov regularization.

SL.polymars : MARS is a non-parametric regression procedure that is used to predict values of continuous outcome variable from predictor variables whose underlying functional relationship (e.g., logistic, linear, etc.) is not assumed. It constructs basis functions and coefficients of relationship entirely from the regression data.

SL.rpartPrune: This is a version of SL.rpart with built-in pruning. SL.rpart is from the the rpart packagefor Recursive Partitioning and Regression Trees. This algorithm builds regression (in our case) or classification models using a two stage model where the resulting models are represented as binary trees. In the first stage, one variable is chosen which best splits the data into two groups. After the data is separated, this process is applied separately to each group in a recursive way until subgroups reach some minimum size or no improvement can be made. In the second stage, cross-validation is used to trim back the full tree. With pruning, we prune back the tree to avoid overfitting.

SL.mean: This is the weighted arithmetic mean function.

4.4 Create data frame with predictor variables. Include predictor variables and transformed variables.

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

> head(X)

W1 W2 W3 W4 W5 Fold sinW3 W4sq

1 0 1 1.3154797 0.3032379 1 1 0.96758337 0.09195322

2 0 1 4.0185981 0.6308452 3 1 -0.76882743 0.39796567

3 1 1 0.1646284 0.6374620 1 1 0.16388578 0.40635782

4 0 1 3.1764225 0.2709290 2 1 -0.03482277 0.07340254

5 0 0 2.8301433 0.3358813 3 1 0.30643860 0.11281624

6 0 1 2.5680567 0.5695696 1 1 0.54260563 0.32440958

cosW5

1 0.5403023

2 -0.9899925

3 0.5403023

4 -0.4161468

5 -0.9899925

6 0.5403023

4.5 Run SuperLearner

> SL.out<- SuperLearner(Y=ObsData$Y, X=X, SL.library=SL.library, cvControl=list(V=20))

Loading required package: polspline

Loading required package: MASS

Loading required package: rpart

Warning message:

package ‘polspline’ was built under R version 2.15.3

4.6 Output

> SL.out

Call:

SuperLearner(Y = ObsData$Y, X = X, SL.library = SL.library, cvControl = list(V = 20))

Risk Coef

SL.glm.EstA\_All 8.312171 0.00000000

SL.glm.EstB\_All 13.022424 0.00000000

SL.glm.EstC\_All 7.763466 0.22829199

SL.glm.EstD\_All 16.036930 0.03487636

SL.ridge\_All 3.995689 0.48465940

SL.rpartPrune\_All 4.807167 0.25217225

SL.polymars\_All 28.293919 0.00000000

SL.mean\_All 123.816749 0.00000000

Running this a second time. Here are the (slightly different) results:

> SL.out<- SuperLearner(Y=ObsData$Y, X=X, SL.library=SL.library, cvControl=list(V=20))

> SL.out

Call:

SuperLearner(Y = ObsData$Y, X = X, SL.library = SL.library, cvControl = list(V = 20))

Risk Coef

SL.glm.EstA\_All 8.312781 0.00000000

SL.glm.EstB\_All 13.016676 0.00000000

SL.glm.EstC\_All 7.771461 0.22770405

SL.glm.EstD\_All 16.021597 0.03541691

SL.ridge\_All 4.001095 0.48552594

SL.rpartPrune\_All 4.808885 0.25135310

SL.polymars\_All 46.661895 0.00000000

SL.mean\_All 123.799381 0.00000000

5.6 Explain Output

Risk – for each algorithm, this is the cross-validated risk, averaged across the 20 folds

Coef – this is the relative weight assigned to each algorithm for the convex combination. Note that algorithms with higher risk generally get lower weights (and conversely those with lower risk get higher weights).

Are cross-validated risks from SuperLearner close to those obtained by my code? Yes!

> colMeans(CV.risk)

[1] 8.312289 13.025325 7.762348 16.031240

5. Implement CV.SuperLearner

5.1 Why we need CV.SuperLearner – note above where I ran Super Learner twice. It gave me slightly different results each time for the risk estimates and weights. Super learner needs to be externally validated to calculate an honest risk for the super learner

5.2 Run CV.SuperLearner

CV.SL.out <- CV.SuperLearner(Y = ObsData$Y, X = X, V = 20, SL.library = SL.library, cvControl = list(V = 20))

5.3 Explore Output

> summary(CV.SL.out)

Call:

CV.SuperLearner(Y = ObsData$Y, X = X, V = 20, SL.library = SL.library,

cvControl = list(V = 20))

Risk is based on: Mean Squared Error

All risk estimates are based on V = 20

Algorithm Ave se Min Max

Super Learner 2.5778 0.058540 1.657536 5.5501

Discrete SL 4.0002 0.073308 3.319975 4.8574

SL.glm.EstA\_All 8.3118 0.218939 6.455879 11.0722

SL.glm.EstB\_All 13.0169 0.242022 11.520813 14.7248

SL.glm.EstC\_All 7.7609 0.219504 6.395013 9.6071

SL.glm.EstD\_All 16.0297 0.268699 14.224486 18.9200

SL.ridge\_All 4.0002 0.073308 3.319975 4.8574

SL.rpartPrune\_All 4.8416 0.109607 3.775555 5.6658

SL.polymars\_All 31.8778 1.012035 0.009239 99.4565

SL.mean\_All 123.8266 1.468596 112.458635 131.7854

CV.SL.out$AllSL

Comment: Most of the algorithms had a relatively constant risk estimate. However, polymars displayed substantial variation across folds. It varied from 10 to 39, depending on the validation fold.

> CV.SL.out$coef

Comment: This shows the coefficients for each algorithm in each fold. For example, SL.gam got the highest coefficient, SL.rpartprune the second highest, etc. The algorithms glm.EstA, glm.EstB, and SL.mean received coefficients of 0. SL.polymars had the most variable coefficient (as expected).

CV.SL.out$whichDiscrete

Comment: For every fold, SL.ridge had the lowest cross-validated risk. I find it interesting that this algorithm was given a coefficient of 0 in the convex combination. I didn’t know that the algorithm with lowest CV risk may not even have a weight in the Super Learner combination.

6 Bonus

6.1 Adding more algorithms to the SuperLearner library

I added SL.gam

SL.library<- c('SL.glm.EstA', 'SL.glm.EstB','SL.glm.EstC', 'SL.glm.EstD', 'SL.ridge','SL.rpartPrune', 'SL.polymars', 'SL.mean', 'SL.gam')

Here, I run CV.SuperLearner

> CV.SL.out <- CV.SuperLearner(Y = ObsData$Y, X = X, V = 20, SL.library = SL.library, cvControl = list(V = 20))

Loading required package: gam

Loading required package: splines

Loaded gam 1.09

Loading required package: polspline

Loading required package: MASS

Loading required package: rpart

Warning messages:

1: package ‘gam’ was built under R version 2.15.3

2: package ‘polspline’ was built under R version 2.15.3

> summary(CV.SL.out)

Call:

CV.SuperLearner(Y = ObsData$Y, X = X, V = 20, SL.library = SL.library,

cvControl = list(V = 20))

Risk is based on: Mean Squared Error

All risk estimates are based on V = 20

Algorithm Ave se Min

Super Learner 2.3445 0.050559 1.821005

Discrete SL 3.9950 0.073050 3.400654

SL.glm.EstA\_All 8.3108 0.218913 6.031421

SL.glm.EstB\_All 13.0241 0.241986 10.764304

SL.glm.EstC\_All 7.7611 0.219607 5.899597

SL.glm.EstD\_All 16.0306 0.268466 13.398325

SL.ridge\_All 3.9950 0.073050 3.400654

SL.rpartPrune\_All 4.8990 0.113090 3.902525

SL.polymars\_All 21.1569 0.886647 0.009303

SL.mean\_All 123.8095 1.468338 114.201875

SL.gam\_All 3.9978 0.073090 3.406668

Max

3.2233

4.4373

9.9323

14.2576

9.4372

18.2750

4.4373

6.6072

106.4373

134.3432

4.4394

Note here that CV.SuperLearner (with gam) does a little bit better for both risk and standard error than the previous computation (without gam). I have pasted the CV.SuperLearner risk estimates from above:

Algorithm Ave se Min Max

Super Learner 2.5778 0.058540 1.657536 5.5501

One could imagine that including every regression algorithm on earth would improve the performance of SuperLearner in predicting with lower risk estimates. However, there is a cost for adding more algorithms. I initially added many algorithms to SL.library (I had to add some extra R packages to use these algorithms)

install.packages (c("glmnet","randomForest", "class", "gam", "gbm", "nnet", "polspline", "MASS", "e1071", "stepPlr", "arm", "party", "spls", "LogicReg", "nnls", "multicore", "SIS", "BayesTree", "quadprog", "ipred", "mlbench", "rpart", "caret", "mda", "earth"), type ="source", repos ="http://cran.cnr.Berkeley.edu", dependencies =c("Depends", "Imports"))

SL.library<- c('SL.glm.EstA', 'SL.glm.EstB','SL.glm.EstC', 'SL.glm.EstD', 'SL.ridge','SL.rpartPrune', 'SL.polymars', 'SL.mean', 'SL.gam', 'SL.gbm', 'SL.glm','SL.glmnet','SL.randomForest','SL.ridge', 'SL.svm','SL.step','SL.step.interaction','SL.bart', 'SL.caret','SL.caret.rpart','SL.earth','SL.loess')

CV.SL.out <- CV.SuperLearner(Y = ObsData$Y, X = X, V = 20, SL.library = SL.library, cvControl = list(V = 20))

However, my code ran from one early evening through the next morning. For some reason, my computer crashed and restarted. Now I remember Anna Decker (in her class presentation) saying that to run SuperLearner it took a couple of weeks. It will be useful – in the future - to figure out how to run this R code on an Amazon EC2 cluster. But I also see why we limit the number of algorithms we choose in practice because of computing time and resources.

6.2 Write a wrapper function

###################

# Estimator E!

SL.glm.EstE<- function(Y, X, newX, family, ...) {

if(family$family=='binomial') {}

if(family$family=='gaussian'){

fit.glm<- glm(Y~ W1\*W2\*W3\*W4\*W5\*sinW3\*W4sq\*cosW5, data=X, family=family)

pred <- predict(fit.glm, newdata=newX, type='response')

fit<- list(object=fit.glm)

}

out <- list(pred=pred, fit=fit)

class(out$fit) <- c('SL.glm.EstE')

return(out)

}

###################

SL.gam.2 <- function (..., deg.gam = 2) {

SL.gam (... , deg.gam=deg.gam )

}