Tutorial 5: Regression II: Model Selection

ECO3080: Machine Learning in Business

Instructor: Prof. Qihui Chen
Teaching Assistant: Long Ma

October 10, 2022

Road Map Today

1. Resampling methods:
   - Leave-one-out cross validation (LOOCV)
   - k-fold cross validation (k-fold CV)
   - Bootstrap
2. Model Selection:
   - Best subset, Forward selection, Backward selection
   - Ridge/Lasso/Elastic net regression
   - Principal component regression (PCR), Partial least square (PLS)

Road Map Today

- Although we talk about these methods in a regression context, we can use them in classification problems or in more general cases.
- Keep in mind that machine learning methods are not isolated from each other, and please use them flexibly.

- What we are trying to do is:
  1. Use data set Auto.
  2. Run the regression "mpg ∼ horsepower" on each training set.
  3. Make prediction on each validation set.
  4. What are the training sets and validation sets?
- We need to install the package "boot" and run the following code.

```
# LOOCV
install.packages("boot")
library(boot)

glm.fit <- glm(mpg ~ horsepower, data = Auto) # define the model
cv.err <- cv.glm(Auto, glm.fit) # do LOOCV
cv.err$delta # MSE on test set

cv.error <- rep(0, 5) # you can run the above code for different polys
for (i in 1:5) {
  glm.fit <- glm(mpg ~ poly(horsepower, i), data = Auto)
  cv.error [i] <- cv.glm(Auto, glm.fit)$delta[1]
}
cv.error # see the change when the model is growing nonlinear
```

- You will get two numbers: 24.23151, 24.23114
- Output "delta" is a vector of length two. The first component is the raw cross-validation estimate of prediction error. The second component is the adjusted cross-validation estimate. The adjustment is designed to compensate for the bias introduced by not using leave-one-out cross-validation.
- We compare 5 models using for loop and take the first component out of "delta". We get: 24.23151, 19.24821, 19.33498, 19.42443, 19.03321, which means that including the quadratic term into our regression model can drastically reduce the prediction error, but the marginal effect of including higher order terms is quite small.

- The logic behind k-fold CV is the same as that behind LOOCV.

```
> library(boot)
> set.seed(911)
>
> cv.error.10 <- rep(0, 10) # do the same thing here
> for (i in 1:10) {
+     glm.fit <- glm(mpg ~ poly(horsepower, i, raw = TRUE), data = Auto)
+     cv.error.10 [i] <- cv.glm(Auto, glm.fit, K = 10)$delta[1]
+ }
> cv.error.10
 [1] 24.24574 19.54000 19.37452 19.59709 19.60597 19.08630 18.64244 18.65313 18.72918
[10] 19.66931
```

- You can draw a graph to show the relationship between test errors and the number of higher order terms you want to include into you model. (Show in class)

- Advantage of Bootstrap: generate a "distribution" (and hence "variance") of estimators without any assumptions.

```
> boot.fn <- function(data, index){
+     return(coef(lm(mpg ~ horsepower, data = data, subset = index)))
+ }
> library(boot)
> set.seed(911)
> boot(Auto, boot.fn, 1000)

ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = Auto, statistic = boot.fn, R = 1000)


Bootstrap Statistics :
      original       bias    std. error
t1* 39.9358610  0.06029772  0.86456731
t2* -0.1578447 -0.00054708  0.00748468
> summary(lm(mpg ~ horsepower, data = Auto))$coef
              Estimate  Std. Error   t value      Pr(>|t|)
(Intercept) 39.9358610 0.717498656  55.65984 1.220362e-187
horsepower  -0.1578447 0.006445501 -24.48914  7.031989e-81
```

- You need to install the package "leaps".

```
library(ISLR)
Hitters <- na.omit(Hitters) # drop those observations with n.a.

install.packages("leaps")
library(leaps)
regfit.full <- regsubsets(Salary ~ ., Hitters, nvmax = 5)
summary(regfit.full)
regfit.fwd <- regsubsets(Salary ~ ., Hitters, nvmax = 5, method = "forward")
summary(regfit.fwd)
regfit.bwd <- regsubsets(Salary ~ ., Hitters, nvmax = 5, method = "backward")
summary(regfit.bwd)
```

- "nvmax" means the maximum number of variables you want to include into your model.
- "method" can be "forward" and "backward". If you do not announce it, then it will use the best subset method.

- How to read the outcome?

```
Selection Algorithm: forward
         AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits CHmRun CRuns CRBI
1  ( 1 ) " "   " "  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"
2  ( 1 ) " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"
3  ( 1 ) " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"
4  ( 1 ) " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"
5  ( 1 ) "*"   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"
         CWalks LeagueN DivisionW PutOuts Assists Errors NewLeagueN
1  ( 1 ) " "    " "     " "       " "     " "     " "    " "
2  ( 1 ) " "    " "     " "       " "     " "     " "    " "
3  ( 1 ) " "    " "     " "       "*"     " "     " "    " "
4  ( 1 ) " "    " "     "*"       "*"     " "     " "    " "
5  ( 1 ) " "    " "     "*"       "*"     " "     " "    " "
```

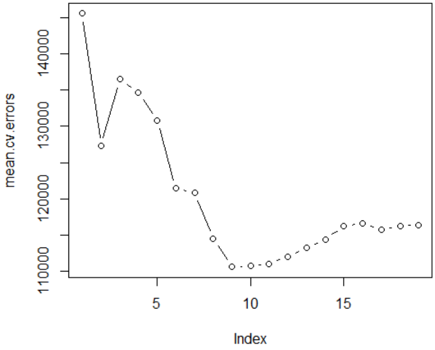
- Each line tells you that which variable should be included into your model when you only need 1 regressor, 2 regressors, and so on.

- Use k-fold CV to find the best model.

```
# define a function of prediction under regsubsets
predict.regsubsets <- function(object, newdata, id) {
  form <- as.formula(object$call[[2]])
  mat <- model.matrix(form, newdata)
  coefi = coef(object, id = id)
  xvars <- names(coefi)
  mat[, xvars] %*% coefi
}

k = 10
set.seed(911)
folds <- sample(1:k, nrow(Hitters), replace = TRUE)
cv.error <- matrix(NA, k, 19, dimnames = list(NULL, paste(1:19)))

for (j in 1:k) {
  best.fit <- regsubsets(Salary ~ ., data = Hitters[folds != j, ], nvmax = 19)
  for (i in 1:19){
    pred <- predict.regsubsets(best.fit, Hitters[folds == j, ], id = i)
    cv.error[j, i] <- mean((Hitters$Salary[folds == j] - pred)^2)
  }
}

mean.cv.errors <- apply(cv.error, 2, mean)
par(mfrow = c(1, 1))
plot(mean.cv.errors, type = "b")
```

Model Selection: BS/F/B

- Use k-fold CV to find the best model.



- General form:

$$\min_{\beta} \sum_{i=1}^{N} \left(y_i - \sum_{j=1}^{K} x_{i,j}\beta_j \right)^2 + \lambda \left[\alpha \|\beta\| + (1-\alpha) \|\beta\|^2 \right] \tag{1}$$

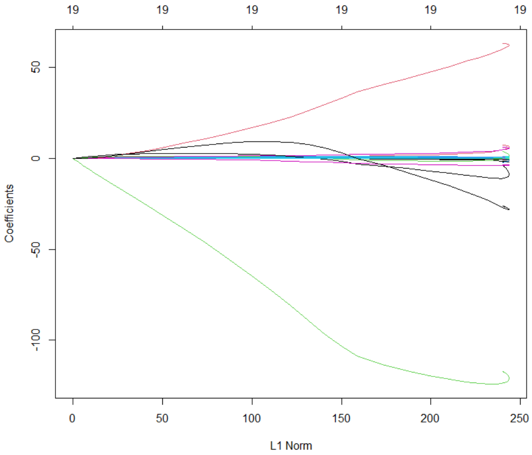
  - When $\alpha = 0$, ridge regression;
  - When $\alpha = 1$, lasso regression;
  - When $\alpha \in (0,1)$, elastic net regression.
- What about the objective function in logistic regression with penalty?

Model Selection: R/L/EN

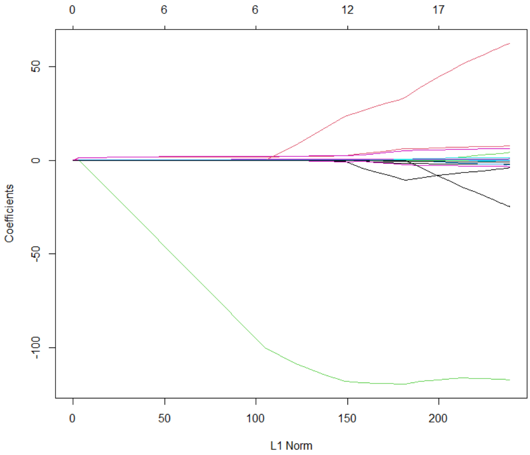
- You need to install the package "glmnet".
- Variables should be in matrix or vector form.
- Variables should be standardized.

```
# ridge regression
ridge.mod <- glmnet(X, Y, alpha = 0, lambda = 5)
summary(ridge.mod)
ridge.mod$beta
ridge.mod$a0

# lasso regression
lasso.mod <- glmnet(X, Y, alpha = 1, lambda = 5)
summary(lasso.mod)
lasso.mod$beta
lasso.mod$a0

# elastic net regression
elasticnet.mod <- glmnet(X, Y, alpha = 0.5, lambda = 5)
summary(elasticnet.mod)
elasticnet.mod$beta
elasticnet.mod$a0
```

- Coefficients shrinkage in ridge regression:

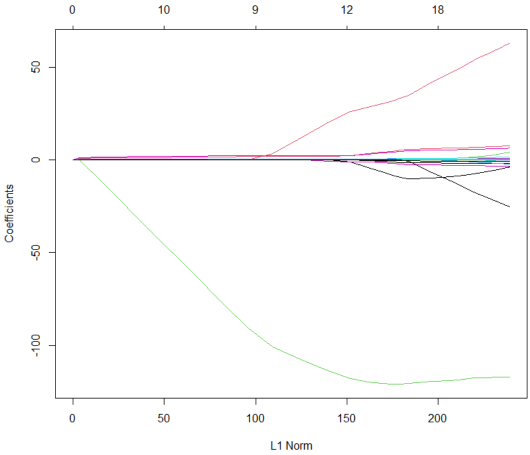


- Coefficients shrinkage in lasso regression:



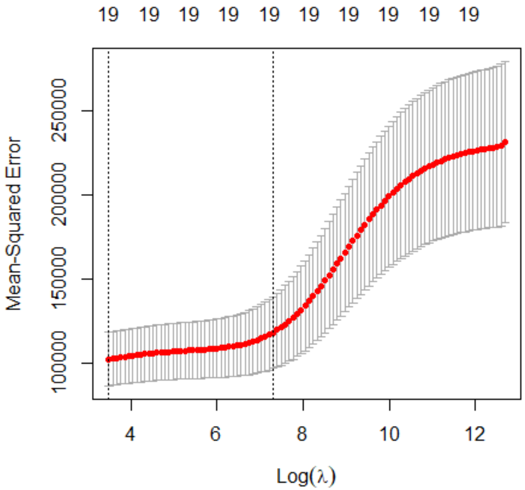
Model Selection: R/L/EN

- Coefficients shrinkage in elastic net regression:



- How to find the best λ and α? k-fold CV.
- We have something called "cv.glmnet".
- Take ridge regression as an example.

```
set.seed(1997)
train <- sample(1:nrow(X), nrow(X)/2)
test <- (-train)
Y.test <- Y[test]

cv.out <- cv.glmnet(X[train,], Y[train], alpha = 0)
plot(cv.out)
bestlambda <- cv.out$lambda.min
bestlambda

ridge.pred <- predict(ridge.mod, s = bestlambda, newx = X[test,])
mean((ridge.pred - Y.test)^2)

out <- glmnet(X, Y, alpha = 0)
predict(out, type = "coefficients", s = bestlambda)[1:20,]
```

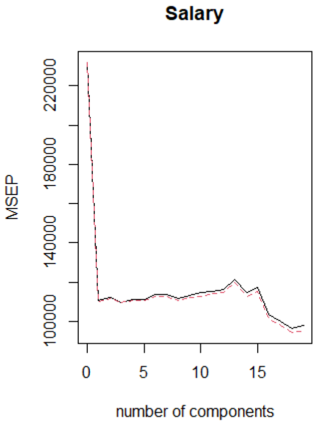

19 19 19 19 19 19 19 19 19 19
Mean-Squared Error
100000
150000
200000
250000
4
6
8
10
12
Log(λ)

- How to choose the best α in elastic net regression?
- Usually, you can use a for loop on a grid of α (e.g. $\alpha \in \{0, 0.1, 0.2, 0.3, \cdots, 0.9, 1\}$)
- The trade-off here is between the prediction power and the computational power.

- Variables should be standardized (R will do it automatically).
- You need to install the package ”pls”.

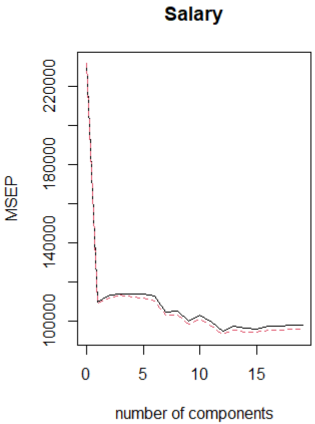
```
set.seed(123)
pcr.fit <- pcr(Salary ~ ., data = Hitters, subset = train,
               scale = TRUE, validation = "CV")
validationplot(pcr.fit, val.type = "MSEP")

pcr.pred <- predict(pcr.fit, X[test, ], ncomp = 7)
mean((pcr.pred - Y.test)^2)

set.seed(123)
pls.fit <- plsr(Salary ~ ., data = Hitters, subset = train,
               scale = TRUE, validation = "CV")
summary(pls.fit)
validationplot(pls.fit, val.type = "MSEP")

pls.pred <- predict(pls.fit, X[test, ], ncomp = 2)
mean((pls.pred - Y.test)^2)
```

- PCR:



Salary
MSEP
100000
140000
180000
220000
0
5
10
15
number of components

- PLS:



Salary
MSEP
100000
140000
180000
220000
0
5
10
15
number of components