Cross Validation and Bootstrap Ch. 5 Exercises

Lucy L.

2024-01-27

```
library(ISLR2)
library(boot)
library(MASS)
```

```
##
## Attaching package: 'MASS'

## The following object is masked from 'package:ISLR2':
##
##     Boston
```

```
set.seed(1)
```

Exercise 5

In Chapter 4, we used logistic regression to predict the probability of `default` using `income` and `balance` on the `Default` data set. We will now estimate the test error of this logistic regression model using the validation set approach. Do not forget to set a random seed before beginning your analysis.

(a) Fit a logistic regression model that uses income and balance to predict `default`.

```
log_mod = glm(default ~ income + balance, data = Default, family = binomial)
attach(Default)
```

(b) Using the validation set approach, estimate the test error of this model.

```
# split into training and validation sets
train_num = 1:5000
train = Default[train_num, ]
validation = Default[-train_num, ]

# generate logistic regression model using training set
log_train = glm(default ~ income + balance, data = train, family = binomial)

# generate predictions using test set
preds1 = predict(log_train, validation, type = "response")
glm.pred1 = rep(0, 5000)
glm.pred1[preds1 > 0.5] = "Yes"
glm.pred1[preds1 <= 0.5] = "No"
table(glm.pred1, validation$default)
```

```
##
## glm.pred1   No  Yes
##       No  4819  106
##       Yes   23   52
```

```
# calculate test error
paste("Logistic regression test error: ",
      1 - mean((glm.pred1 == validation$default)))
```

```
## [1] "Logistic regression test error:  0.0258"
```

(c) Repeat the process in (b) three times, using three different splits of the observations into a training set and a validation set. Comment on the results obtained.

```
split_1 = 1:3000
split_2 = 1:6000
split_3 = 1:8000

train1 = Default[split_1, ]
validation1 = Default[-split_1, ]
train2 = Default[split_2, ]
validation2 = Default[-split_2, ]
train3 = Default[split_3, ]
validation3 = Default[-split_3, ]

log_train1 = glm(default ~ income + balance, data = train1, family = binomial)
preds11 = predict(log_train1, validation1, type = "response")
glm.pred11 = rep(0, 7000)
glm.pred11[preds11 > 0.5] = "Yes"
glm.pred11[preds11 <= 0.5] = "No"
table(glm.pred11, validation1$default)
```

```
##
## glm.pred11   No  Yes
##        No  6728  145
##        Yes   40   87
```

```
log_train2 = glm(default ~ income + balance, data = train2, family = binomial)
preds12 = predict(log_train2, validation2, type = "response")
glm.pred12 = rep(0, 4000)
glm.pred12[preds12 > 0.5] = "Yes"
glm.pred12[preds12 <= 0.5] = "No"
table(glm.pred12, validation2$default)
```

```
##
## glm.pred12   No  Yes
##        No  3852   90
##        Yes   14   44
```

```
log_train3 = glm(default ~ income + balance, data = train3, family = binomial)
preds13 = predict(log_train3, validation3, type = "response")
glm.pred13 = rep(0, 2000)
glm.pred13[preds13 > 0.5] = "Yes"
glm.pred13[preds13 <= 0.5] = "No"
table(glm.pred13, validation3$default)
```

```
##
## glm.pred13   No  Yes
##        No  1926   43
##        Yes    7   24
```

```
paste("Split 1 Logistic regression test error: ",
      1 - mean((glm.pred11 == validation1$default)))
```

```
## [1] "Split 1 Logistic regression test error:  0.0264285714285715"
```

```
paste("Split 2 Logistic regression test error: ",
      1 - mean((glm.pred12 == validation2$default)))
```

```
## [1] "Split 2 Logistic regression test error:  0.026"
```

```
paste("Split 3 Logistic regression test error: ",
      1 - mean((glm.pred13 == validation3$default)))
```

```
## [1] "Split 3 Logistic regression test error:  0.025"
```

The error rate generally ranges between 0.025-0.027 across training splits.

(d) Now consider a logistic regression model that predicts the probability of `default` using `income`, `balance`, and a dummy variable for `student`. Estimate the test error for this model using the validation set approach. Comment on whether or not including a dummy variable for student leads to a reduction in the test error rate.

```
train_num = 1:5000
train = Default[train_num, ]
validation = Default[-train_num, ]

three_pred_mod = glm(default ~ ., data = train, family = binomial)

preds_d = predict(three_pred_mod, validation, type = "response")
glm.pred = rep(0, 5000)
glm.pred[preds_d > 0.5] = "Yes"
glm.pred[preds_d <= 0.5] = "No"
table(glm.pred, validation$default)
```

```
##
## glm.pred   No  Yes
##      No  4816  105
##      Yes   26   53
```

```r
# calculate test error
paste("Logistic regression test error: ",
      1 - mean((glm.pred == validation$default)))
```

```
## [1] "Logistic regression test error:  0.0262"
```

Adding the dummy variable for `student` slightly increased the error rate.

Exercise 6

We continue to consider the use of a logistic regression model to predict the probability of default using income and balance on the `Default` data set. In particular, we will now compute estimates for the standard errors of the `income` and `balance` logistic regression coeffcients in two diferent ways: (1) using the bootstrap, and (2) using the standard formula for computing the standard errors in the `glm()` function. Do not forget to set a random seed before beginning your analysis.

(a) Using the `summary()` and `glm()` functions, determine the estimated standard errors for the coefcients associated with income and balance in a multiple logistic regression model that uses both predictors.

```r
summary(log_mod)
```

```
##
## Call:
## glm(formula = default ~ income + balance, family = binomial,
##     data = Default)
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.154e+01  4.348e-01 -26.545  < 2e-16 ***
## income       2.081e-05  4.985e-06   4.174 2.99e-05 ***
## balance      5.647e-03  2.274e-04  24.836  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2920.6  on 9999  degrees of freedom
## Residual deviance: 1579.0  on 9997  degrees of freedom
## AIC: 1585
##
## Number of Fisher Scoring iterations: 8
```

```r
log_mod$coefficients
```

```
##   (Intercept)        income       balance
## -1.154047e+01  2.080898e-05  5.647103e-03
```

(b) Write a function, `boot.fn()`, that takes as input the `Default` data set as well as an index of the observations, and that outputs the coefficient estimates for income and balance in the multiple logistic regression model.

```r
boot.fn = function(dataset, index) {
  # set training, test sets and logistic regression model
  train = dataset[index, ]
  validation = dataset[-index, ]
  model = glm(default ~ income + balance, data = train, family = binomial)

  return(sqrt(diag(vcov(model))))
}
```

(c) Use the `boot()` function together with your `boot.fn()` function to estimate the standard errors of the logistic regression coefficients for `income` and `balance`.

```r
boot.fn(Default, sample(500, 500, replace = T))
```

```
##  (Intercept)       income      balance
## 1.695066e+00 2.101318e-05 9.399382e-04
```

```r
boot(Default, boot.fn, 500)
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Default, statistic = boot.fn, R = 500)
##
##
## Bootstrap Statistics :
##         original       bias     std. error
## t1* 4.347564e-01 2.029749e-03 2.167373e-02
## t2* 4.985167e-06 1.553626e-08 1.383315e-07
## t3* 2.273731e-04 1.204925e-06 1.111664e-05
```

Exercise 7

In Sections 5.3.2 and 5.3.3, we saw that the `cv.glm()` function can be used in order to compute the LOOCV test error estimate. Alternatively, one could compute those quantities using just the `glm()` and `predict.glm()` functions, and a for loop. You will now take this approach in order to compute the LOOCV error for a simple logistic regression model on the Weekly data set. Recall that in the context of classification problems, the LOOCV error is given in (5.4).

(a) Fit a logistic regression model that predicts `Direction` using `Lag1` and `Lag2`.

```r
weekly = Weekly
simple_mod = glm(Direction ~ Lag1 + Lag2, data = Weekly, family = binomial)
```

(b) Fit a logistic regression model that predicts `Direction` using `Lag1` and `Lag2` using all but the first observation.

```
first_obs = Weekly[1, ]
all_but_first = Weekly[-1, ]
second_simple = glm(Direction ~ Lag1 + Lag2, data = all_but_first,
                    family = binomial)
```

(c) Use the model from (b) to predict the direction of the first observation. You can do this by predicting that the first observation will go up if `P(Direction = "Up"|Lag1, Lag2) > 0.5`. Was this observation correctly classified?

```
single_prediction = predict(second_simple, first_obs, type = "response")
pred_item = rep(0, 1)
pred_item[single_prediction > 0.5] = "Up"
pred_item[single_prediction <= 0.5] = "Down"

pred_item == Weekly[1, 9]
```

```
## [1] FALSE
```

The observation was not correctly classified.

(d) Write a for loop from `i = 1` to `i = n`, where `n` is the number of observations in the data set, that performs each of the following steps: (i.) Fit a logistic regression model using all but the ith observation to predict `Direction` using `Lag1` and `Lag2`. (ii.) Compute the posterior probability of the market moving up for the ith observation. (iii.) Use the posterior probability for the ith observation in order to predict whether or not the market moves up. (iv.) Determine whether or not an error was made in predicting the direction for the ith observation. If an error was made, then indicate this as a 1, and otherwise indicate it as a 0.

```
num_rows = nrow(Weekly)
for (i in 1:num_rows) {
  # Fit model using all but observation i
  observation = Weekly[i, ]
  all_but_observation = Weekly[-i, ]
  model = glm(Direction ~ Lag1 + Lag2, data = all_but_observation,
                    family = binomial)

  # Compute posterior probability for observation i
  prediction = predict(model, observation, type = "response")
  pred_item = rep(0, 1)
  pred_item[prediction > 0.5] = "Up"
  pred_item[prediction <= 0.5] = "Down"

  actual_value = Weekly[i, 9]

  # Record whether prediction was correct
  if (pred_item == actual_value) {
    weekly$loocv_estimate[i] = 0
  } else{
    weekly$loocv_estimate[i] = 1
  }
}
```

(e) Take the average of the n numbers obtained in (d)iv in order to obtain the LOOCV estimate for the test error. Comment on the results.

```
mean(weekly$loocv_estimate)
```

```
## [1] 0.4499541
```

The LOOCV estimate for the test error is about 0.449.

Exercise 8

We will now perform cross-validation on a simulated data set.

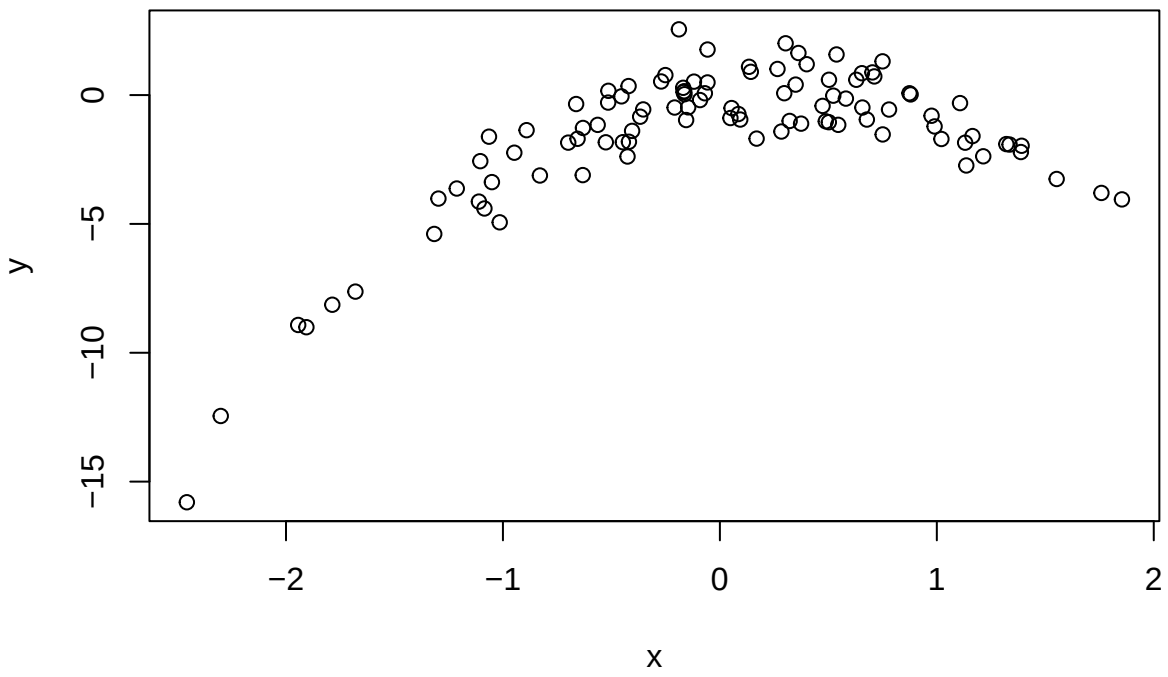
(a) Generate a simulated data set as follows. In this data set, what is `n` and what is `p`? Write out the model used to generate the data in equation form.

```
x = rnorm(100)
y = x - 2 * x^2 + rnorm(100)
```

`n` is 100 and `p` is 2. The model is $Y = X - 2X^2 + \epsilon$.

(b) Create a scatterplot of `X` against `Y` . Comment on what you fnd.

```
plot(x, y)
```



We observe a quadratic relationship between the predictors and the response.

(c) Set a random seed, and then compute the LOOCV errors that result from fitting the following four models using least squares.

```
# Generate models
combined = data.frame(x, y)
mod_1 = glm(y ~ x, data = combined)
mod_2 = glm(y ~ poly(x, 2), data = combined)
mod_3 = glm(y ~ poly(x, 3), data = combined)
mod_4 = glm(y ~ poly(x, 4), data = combined)

# LOOCV errors
cv.glm(combined, mod_1)$delta[1]
```

```
## [1] 5.95213
```

```
cv.glm(combined, mod_2)$delta[1]
```

```
## [1] 0.9011052
```

```
cv.glm(combined, mod_3)$delta[1]
```

```
## [1] 0.9125909
```

```
cv.glm(combined, mod_4)$delta[1]
```

```
## [1] 0.9490138
```

(d) Repeat (c) using another random seed, and report your results. Are your results the same as what you got in (c)? Why?

```
set.seed(2)

x = rnorm(100)
y = x - 2 * x^2 + rnorm(100)

# Generate models
combined = data.frame(x, y)
mod_1 = glm(y ~ x, data = combined)
mod_2 = glm(y ~ poly(x, 2), data = combined)
mod_3 = glm(y ~ poly(x, 3), data = combined)
mod_4 = glm(y ~ poly(x, 4), data = combined)

# LOOCV errors
cv.glm(combined, mod_1)$delta[1]
```

```
## [1] 9.858301
```

```
cv.glm(combined, mod_2)$delta[1]
```

```
## [1] 1.00441
```

```
cv.glm(combined, mod_3)$delta[1]
```

```
## [1] 1.01803
```

```
cv.glm(combined, mod_4)$delta[1]
```

```
## [1] 1.035601
```

The results are different after setting a different seed due to the random error in the data.

(e) Which of the models in (c) had the smallest LOOCV error? Is this what you expected? Explain your answer.

The models with any of the polynomial predictors had the lowest LOOCV errors. This is as expected since the plot for `y` against `x` followed a nonlinear pattern.

(f) Comment on the statistical significance of the coefficient estimates that results from fitting each of the models in (c) using least squares. Do these results agree with the conclusions drawn based on the cross-validation results?

```
summary(mod_1); summary(mod_2); summary (mod_3); summary(mod_4)
```

```
##
## Call:
## glm(formula = y ~ x, data = combined)
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -2.6434     0.3072  -8.606 1.27e-13 ***
## x             0.8181     0.2660   3.076  0.00272 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 9.428005)
##
##     Null deviance: 1013.12  on 99  degrees of freedom
## Residual deviance:  923.94  on 98  degrees of freedom
## AIC: 512.14
##
## Number of Fisher Scoring iterations: 2
```

```
##
## Call:
## glm(formula = y ~ poly(x, 2), data = combined)
##
```

```
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -2.66853    0.09907 -26.936  < 2e-16 ***
## poly(x, 2)1   9.44338    0.99068   9.532 1.37e-15 ***
## poly(x, 2)2 -28.78792    0.99068 -29.059  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.9814425)
##
##     Null deviance: 1013.1  on 99  degrees of freedom
## Residual deviance:   95.2  on 97  degrees of freedom
## AIC: 286.87
##
## Number of Fisher Scoring iterations: 2

##
## Call:
## glm(formula = y ~ poly(x, 3), data = combined)
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -2.66853    0.09956 -26.804  < 2e-16 ***
## poly(x, 3)1   9.44338    0.99558   9.485  1.9e-15 ***
## poly(x, 3)2 -28.78792    0.99558 -28.916  < 2e-16 ***
## poly(x, 3)3  -0.21633    0.99558  -0.217    0.828
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.9911784)
##
##     Null deviance: 1013.122  on 99  degrees of freedom
## Residual deviance:   95.153  on 96  degrees of freedom
## AIC: 288.82
##
## Number of Fisher Scoring iterations: 2

##
## Call:
## glm(formula = y ~ poly(x, 4), data = combined)
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -2.6685     0.1001 -26.665  < 2e-16 ***
## poly(x, 4)1   9.4434     1.0008   9.436 2.65e-15 ***
## poly(x, 4)2 -28.7879     1.0008 -28.766  < 2e-16 ***
## poly(x, 4)3  -0.2163     1.0008  -0.216    0.829
## poly(x, 4)4   0.0866     1.0008   0.087    0.931
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 1.001533)
##
##     Null deviance: 1013.122  on 99  degrees of freedom
```

```
## Residual deviance:   95.146  on 95  degrees of freedom
## AIC: 290.81
##
## Number of Fisher Scoring iterations: 2
```

It appears that only up to X^2 are the predictors statistically significant in predicting `Y`. This does not necessarily disagree with the results from the previous parts because adding additional unnecessary (polynomial) predictors may minimally still help improve the prediction of the model. Instead, this summary provides clarification for which predictors are responsible for the bulk of the low LOOCV errors previously seen.

Exercise 9

We will now consider the `Boston` housing data set, from the `ISLR2` library.

(a) Based on this data set, provide an estimate for the population mean of `medv`. Call this estimate $\hat{mu}$.

```
boston = Boston
(mu_hat = mean(Boston$medv))
```

```
## [1] 22.53281
```

(b) Provide an estimate of the standard error of $\hat{mu}$. Interpret this result.

```
(mu_se = sd(Boston$medv) / sqrt(nrow(Boston)))
```

```
## [1] 0.4088611
```

(c) Now estimate the standard error of $\hat{mu}$ using the bootstrap. How does this compare to your answer from (b)?

```
medv = boston$medv
boot.mu.se = function(d, i) {
  d2 = d[i,]
  return(mean(d2$medv))
}

(boot_mu = boot(boston, boot.mu.se, R = 1000))
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = boston, statistic = boot.mu.se, R = 1000)
##
##
## Bootstrap Statistics :
##     original       bias    std. error
## t1* 22.53281 -0.009336759   0.4074408
```

Using 1000 bootstrap samples, the bootstrap estimate is approximate to the one previously calculated.

(d) Based on your bootstrap estimate from (c), provide a 95 percent confidence interval for the mean of `medv`. Compare it to the results obtained using `t.test(Boston$medv)`.

```
mu_hat - (2 * mu_se)
```

```
## [1] 21.71508
```

```
mu_hat + (2 * mu_se)
```

```
## [1] 23.35053
```

```
t.test(Boston$medv)
```

```
##
##  One Sample t-test
##
## data:  Boston$medv
## t = 55.111, df = 505, p-value < 2.2e-16
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  21.72953 23.33608
## sample estimates:
## mean of x
##  22.53281
```

They are approximate to each other.

(e) Based on this data set, provide an estimate, $\widehat{\mu_{med}}$, for the median value of `medv` in the population.

```
(median_mu = median(Boston$medv))
```

```
## [1] 21.2
```

(f) We now would like to estimate the standard error of $\widehat{\mu_{med}}$. Unfortunately, there is no simple formula for computing the standard error of the median. Instead, estimate the standard error of the median using the bootstrap. Comment on your findings.

```
boot.median = function(d, i) {
  d2 = d[i,]
  return(median(d2$medv))
}

(boot_results =  boot(boston, boot.median, 1000))
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
```

```
##
## Call:
## boot(data = boston, statistic = boot.median, R = 1000)
##
##
## Bootstrap Statistics :
##     original   bias    std. error
## t1*     21.2 -0.02655   0.3746972
```

Using 1000 bootstrap samples, the SE for the median is also approximate to that for the mean.

(g) Based on this data set, provide an estimate for the tenth percentile of `medv` in Boston census tracts. Call this quantity $\hat{\mu_{0.1}}$. (You can use the `quantile()` function.)

```
quantile(boston$medv, probs = 0.1)
```

```
##   10%
## 12.75
```

(h) Use the bootstrap to estimate the standard error of $\hat{\mu_{0.1}}$. Comment on your findings.

```
tenth_percentile = function(d, i) {
  d2 = d[i, ]
  return(quantile(d2$medv, probs = 0.1))
}

boot(boston, tenth_percentile, 1000)
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = boston, statistic = tenth_percentile, R = 1000)
##
##
## Bootstrap Statistics :
##     original   bias    std. error
## t1*    12.75 -0.00845   0.5037011
```

The standard error for the estimate of the 10th percentile of `medv` is about 0.493. Interestingly, this is about 0.1 greater than the bootstrapped SE estimates for the measures of center (mean and median). We may have expected this to be more similar because for each bootstrap measurement, we are technically measuring the SE for the value of `medv` at a certain percentile of `medv` (such 10th percentile, 50th percentile for the median, the percentile corresponding to the mean, etc.). We may also think that this is actually a sensible result because the expected value for `medv` itself may vary across various values of `medv`.