ch-5-answers

Conner McCloney

1 Simulation

```
What if you calculated the test MSE for every possible layout of
k-fold CV, if this were computationally feasible would it improve the
accuracy or precision and if so, by how much? What I mean is, if n =
4, and you do 2-fold CV, you would need to repeat the 2-fold CV
process 2 times, once for every group of k=2 you could form.
```

This is a great question, and it's something you could investigate with simulation. As you mention - it could defeat the computational advantage of doing k-fold CV instead of n-fold cv. Why don't you explore this for a small simulated data set. That is, simulate data under the linear model: $y_i{=}\beta_0{+}\beta_1 x_i{+}\epsilon_i$, where $\epsilon_i \sim$ N(0,σ^2), and i=1,2,3,…,10. Fit the model and use 5-fold CV for every possible combo of folds. Compare that to 5-fold CV on just one randomly selected 5-fold. See what you find. Note that you'll have to pick values for β_0,β_1,and ϵ to do the simulation I'll leave that up to you! Additionally, you could think about the additional computational burdon of finding every possible k-fold CV from n data points by counting the number of different k-fold sets there are from n distinct values.

```
my.cv.glm <- function (data, glmfit, cost = function(y, yhat) mean((y - yhat)^2),
    K = n)
{
    call <- match.call()
    if (!exists(".Random.seed", envir = .GlobalEnv, inherits = FALSE))
        runif(1)
    seed <- get(".Random.seed", envir = .GlobalEnv, inherits = FALSE)
    n <- nrow(data)
    if ((K > n) || (K <= 1))
        stop("'K' outside allowable range")
    K.o <- K
    K <- round(K)
    kvals <- unique(round(n/(1L:floor(n/2))))
    temp <- abs(kvals - K)
    if (!any(temp == 0))
        K <- kvals[temp == min(temp)][1L]
    if (K != K.o)
        warning(gettextf("'K' has been set to %f", K),
            domain = NA)
    f <- ceiling(n/K)
    s <- sample(rep(1L:K, f), n)
    n.s <- table(s)
    glm.y <- glmfit$y
    cost.0 <- cost(glm.y, fitted(glmfit))
    ms <- max(s)
    CV <- 0
    Call <- glmfit$call
    my.temp <- combn(s,n)
    #for (i in seq_len(ms)) {
        #cat("i",i,"\n")
        #j.out <- seq_len(n)[(s == i)]
        #cat("j.out",j.out,"\n")
        #j.in <- seq_len(n)[(s != i)]
        #cat("j.in",j.in,"\n")
    ###Code I added
    c <- combn(glm.y,f)
    l <- length(c[1,])
    for(m in seq_len(l)){
        j.in <- which(glm.y %in% c[,m])
        j.out <- seq_len(n)
        for(i in seq_len(length(j.in))){
          j.out <- j.out[which(j.out!=j.in[i])]
        }
    ###
        Call$data <- data[j.in, , drop = FALSE]
        d.glm <- eval.parent(Call)
        ###changed from p.alpha <- n.s[i]/n (which = 1 / k, I think)
        p.alpha <- 1/(l)
        cost.i <- cost(glm.y[j.out], predict(d.glm, data[j.out,
            , drop = FALSE], type = "response"))
        CV <- (CV + p.alpha * cost.i)
        cost.0 <- cost.0 - p.alpha * cost(glm.y, predict(d.glm,
            data, type = "response"))
    }
    list(call = call, K = K, delta = as.numeric(c(CV, CV + cost.0)), seed = seed)
}
```

```
library(boot)
library(utils)
set.seed(10102019)
k <- 5
n <- 20
beta_0 <- 10
beta_1 <- 2
e.sigma <- 1
epsilon <- rnorm(n, mean = 0, sd = e.sigma)

x.mean <- sample(1:10,1)
x.sd <- sample(1:10,1)
x <- rnorm(n, mean = x.mean, sd = x.sd)

y <- beta_0 + beta_1*x + epsilon

my.data <- data.frame(y,x)
#------------
#.og -> original k-fold CV with k = 5
#.all -> runs k-fold using all possible folds,
#       using the function I modified above

timings.og=rep(0,10)
timings.all=rep(0,10)

cv.error.og=rep(0,10)
cv.error.all=rep(0,10)

for(i in 1:10){
```

```r
  glm.fit=glm(y~x,data=my.data)
  start.time <- Sys.time()
  cv.error.og[i]=cv.glm(my.data,glm.fit,K=k)$delta[2]
  end.time <- Sys.time()
  timings.og[i] <- end.time - start.time

  start.time <- Sys.time()
  cv.error.all[i]=my.cv.glm(my.data,glm.fit,K=k)$delta[2]
  end.time <- Sys.time()
  timings.all[i] <- end.time - start.time

  x <- rnorm(n, mean = x.mean, sd = x.sd)
  y <- beta_0 + beta_1*x + epsilon
  my.data <- data.frame(y,x)
}
cat("-----Results-----\n")
```

```
## -----Results-----
```

```r
cv.error.og
```

```
##  [1] 0.7732017 0.8793351 0.8638289 1.0354212 0.8563259 0.8677541 0.9237342
##  [8] 1.0881843 0.8218131 0.8404049
```

```r
cv.error.all
```

```
##  [1] 0.9083821 0.9100890 0.9683535 1.1708470 0.9523250 0.9416892 0.9630296
##  [8] 0.9162416 1.2659651 0.9440237
```

```r
cat("-----Means-----\n")
```

```
## -----Means-----
```

```r
mean(cv.error.og)
```

```
## [1] 0.8950003
```

```r
mean(cv.error.all)
```

```
## [1] 0.9940946
```

```r
cat("-----Computation Times-----\n")
```

```
## -----Computation Times-----
```

```r
timings.og
```

```
##  [1] 0.011995077 0.008125067 0.011552095 0.015626907 0.008990049
##  [6] 0.008995056 0.010146141 0.010987043 0.000000000 0.002079010
```

```r
timings.all
```

```
##  [1]  9.299614  9.113207  9.161969  9.087248  9.049383  9.049116  9.302429
##  [8] 10.514711  9.556021  9.106872
```

```r
cat("-----Means of Times-----\n")
```

```
## -----Means of Times-----
```

```r
mean(timings.og)
```

```
## [1] 0.008849645
```

```r
mean(timings.all)
```

```
## [1] 9.324057
```

For $n = 20$ values with $k = 5$, a fold will be of size 4, so there are ${}_{20}C_4 = 4,845$ total folds that are being analyzed compared to only $k = 5$ folds for normal k-fold CV, or 969x as many in this case. The timings above reflect this fact as well.

2 Conceptual (2)

a)

What is the probability that the first bootstrap observation is not the jth observation from the original sample? Justify your answer.

The first observation in the bootstrap sample has $\frac{1}{n}$ probability of being any individual observation from the original sample. So, the probability of not being this given observation is $1 - \frac{1}{n}$, or $\frac{n-1}{n}$.

$\Pr(B_1 \mathrel{!}= O_j) = \frac{n-1}{n}$

b)

What is the probability that the second bootstrap observation is not the jth observation from the original sample?

$\Pr(B_2 \mathrel{!}= O_j) = \frac{n-1}{n}$

c)

Argue that the probability that the jth observation is not in the bootstrap sample is $(1 - \frac{1}{n})^n$.

For each bootstrap observation, there is a $\frac{1}{n}$ probability of the jth observation being the ith observation in the bootstrap sample. Because there are n observations, this probability is multiplied together n times for each observation, resulting in the overall probability of not being included as $(1 - \frac{1}{n})^n$.

d)

When n = 5, what is the probability that the jth observation is in the bootstrap sample?

$1 - (1 - \frac{1}{n})^n = 1 - (1 - \frac{1}{5})^5 = 0.67232.$

e)

When n = 100, what is the probability that the jth observation is in the bootstrap sample?

$1 - (1 - \frac{1}{n})^n = 1 - (1 - \frac{1}{100})^{100} = 0.6339677.$

f)

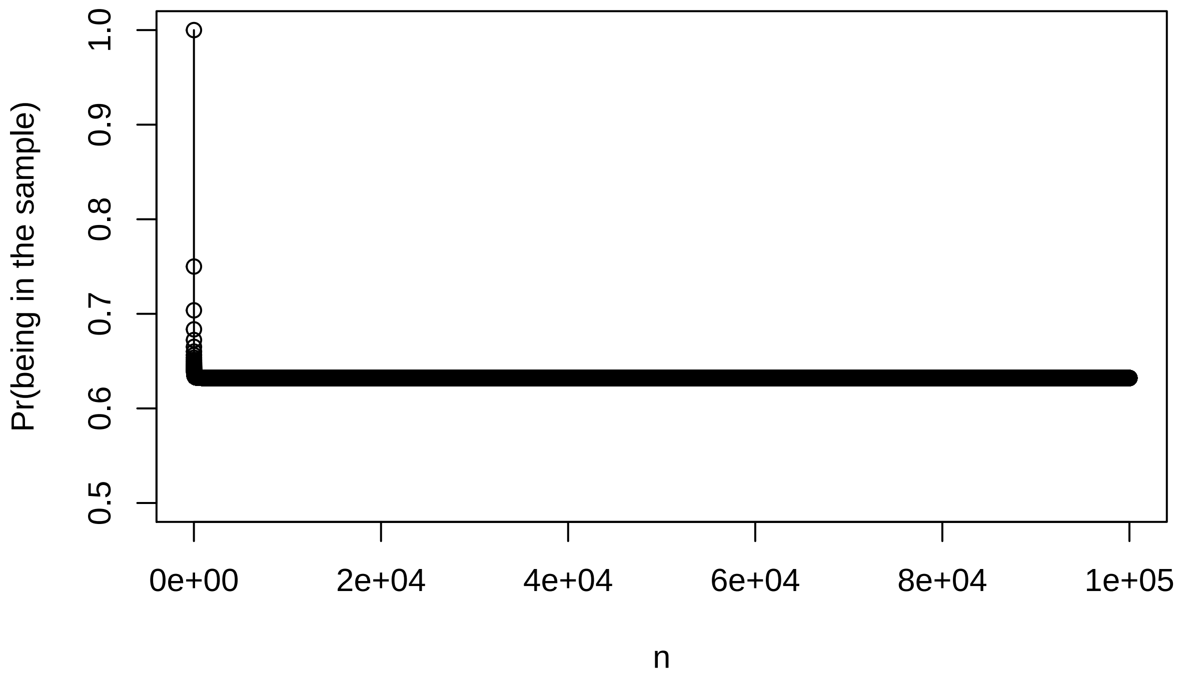
When n = 10, 000, what is the probability that the jth observation is in the bootstrap sample?

$1 - (1 - \frac{1}{n})^n = 1 - (1 - \frac{1}{10000})^{10000} = 0.632139.$

g)

Create a plot that displays, for each integer value of n from 1 to 100, 000, the probability that the jth observation is in the bootstrap sample. Comment on what you observe.

```
n <- 100000
x1 <- seq(1:n)
y1 <- 1 - (1 - (1/x1))^x1
plot(y1~x1,ylim=c(0.5,1),xlab="n",ylab="Pr(being in the sample)")
lines(x1, y1)
```



The probability of the jth observation being included in the sample is 100% for 1 observation, then quickly drops off to ~0.63, before remaining around 0.63 for nearly all n up to and including n = 100,000.

h)

We will now investigate numerically the probability that a bootstrap sample of size n = 100 contains the jth observation. Here j = 4. We repeatedly create bootstrap samples, and each time we record whether or not the fourth observation is contained in the bootstrap sample. Comment on the results obtained.

```
y1[10000] #true prob
```

```
## [1] 0.632139
```

```
store=rep (NA , 10000)
for (i in 1:10000) {
  store[i]=sum(sample (1:100 , rep =TRUE)==4) >0
}
mean(store)
```

```
## [1] 0.625
```

The probability obtained here agrees with the true proability and plots generated from above, with a probability of the j=4th observation being included in the model at around 0.63.

3 Applied (8)

a)

In this data set, what is n and what is p? Write out the model used to generate the data in equation form.

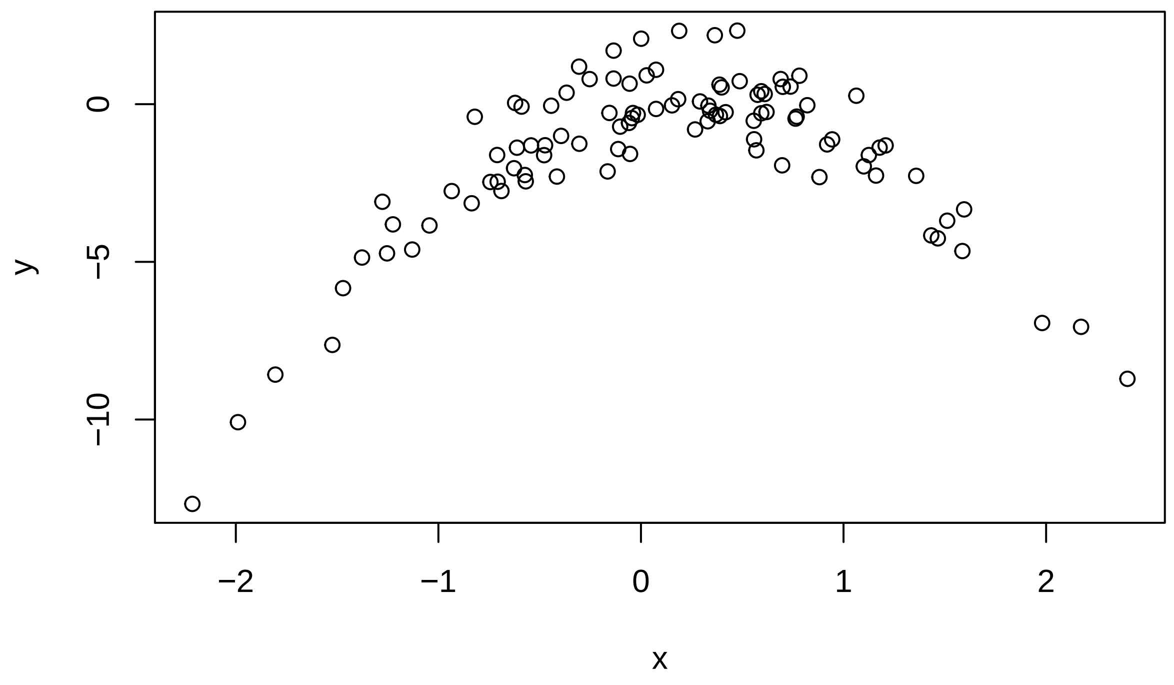
```
set.seed(1)
x=rnorm(100)
y=x-2*x^2+rnorm(100)
```

$y = X - 2X^2 + \epsilon$, where $X \sim N(0,1)$ and $\epsilon \sim N(0,1)$. For this, $n = 100$, and $p = 2$.

b)

Create a scatterplot of X against Y . Comment on what you find.

```
plot(y~x)
```



This plot seems to suggest a curved relationship between X and Y.

c)

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

i. $Y = \beta_0 + \beta_1 X + \epsilon$

ii. $Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \epsilon$

iii. $Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \epsilon$

iv. $Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \beta_4 X^4 + \epsilon$.

Note you may find it helpful to use the data.frame() function to create a single data set containing both X and Y .

```r
set.seed(10112019)

my.data <- data.frame(x,y)

glm.fit1=glm(y~x,data=my.data)
cv.err1=cv.glm(my.data,glm.fit1)
cv.err1$delta
```

```
## [1] 7.288162 7.284744
```

```r
glm.fit2=glm(y~poly(x,2),data=my.data)
cv.err2=cv.glm(my.data,glm.fit2)
cv.err2$delta
```

```
## [1] 0.9374236 0.9371789
```

```r
glm.fit3=glm(y~poly(x,3),data=my.data)
cv.err3=cv.glm(my.data,glm.fit3)
cv.err3$delta
```

```
## [1] 0.9566218 0.9562538
```

```r
glm.fit4=glm(y~poly(x,4),data=my.data)
cv.err4=cv.glm(my.data,glm.fit4)
cv.err4$delta
```

```
## [1] 0.9539049 0.9534453
```

d)

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

```r
library(boot)

set.seed(10122019)

my.data1 <- data.frame(x,y)

glm.fit1=glm(y~x,data=my.data)
```

```
cv.err1=cv.glm(my.data,glm.fit1)
cv.err1$delta
```

```
## [1] 7.288162 7.284744
```

```
glm.fit2=glm(y~poly(x,2),data=my.data)
cv.err2=cv.glm(my.data,glm.fit2)
cv.err2$delta
```

```
## [1] 0.9374236 0.9371789
```

```
glm.fit3=glm(y~poly(x,3),data=my.data)
cv.err3=cv.glm(my.data,glm.fit3)
cv.err3$delta
```

```
## [1] 0.9566218 0.9562538
```

```
glm.fit4=glm(y~poly(x,4),data=my.data)
cv.err4=cv.glm(my.data,glm.fit4)
cv.err4$delta
```

```
## [1] 0.9539049 0.9534453
```

They are the same, because LOOCV evaluates all n folds so the results will always be the same.

e)

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

The second model of Y = β_0 + β_1X + $\beta_2 X^2$ + ϵ had the smallest LOOCV error. I did expect this because the y-values were generated using this model where $\beta_0 = 0$, $\beta_1 = 1$, $\beta_2 = -2$ and $\epsilon \sim N(0,1)$, so this model should have the smallest errors out of the four tested.

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(glm.fit4)$coefficients
```

```
##                  Estimate Std. Error     t value     Pr(>|t|)
## (Intercept)  -1.5500226 0.09590514 -16.1620379 5.169227e-29
## poly(x, 4)1   6.1888256 0.95905143   6.4530695 4.590732e-09
## poly(x, 4)2 -23.9483049 0.95905143 -24.9708243 1.593826e-43
## poly(x, 4)3   0.2641057 0.95905143   0.2753822 7.836207e-01
## poly(x, 4)4   1.2570950 0.95905143   1.3107691 1.930956e-01
```

There are small p-values for only the intercept, linear, and quadratic terms, which are the only terms relevant to the true model. Therefore, these results agree with the conclusions we drew from cross-validation.