5 Cross-validation and bootstrap

ISLR Chapter 5 covers two computational techniques — cross-validation and bootstrap — for model evaluation.

5.1 ISLR 5.1

Cross-validation (CV) can be used for two purposes: model selection and model assessment. In model selection, we select hyperparameters for the final model. In model assessment, we evaluate the generalizability of a model (e.g., prediction performance).

Some issues have been covered in my notes for previous weeks.

- 5.1.1: Validation set approach:
  - Results can vary due to random splitting, especially when n is not large. (Figure 5.2 in comparison with Figure 5.4)
- 5.1.2: Leave-one-out (LOO) CV. It is n-fold CV. It is also called the jackknife.
  - n models are fit. Can be computationally expensive; except for least squares linear regression, for which fitting one model is sufficient. (5.2)
  - Bias is minimal because $n_{train} = n - 1$.
  - Variance may be high because the n models are very similar.
- 5.1.3: k-fold CV
  - Every observation has a chance to serve as a test point.
  - An improvement over 5.1.1, especially for datasets that are not very large.
  - Less computational than 5.1.2. Similar performance on model selection as LOO (Figure 5.6)

Figures 5.6 and 5.8 show that CV curves are good approximations of the true test error curves.

Figure 5.6 is for the simulation settings in Figures 2.9, 2.10, and 2.11. True test MSE vs LOOCV and 10-fold CV.

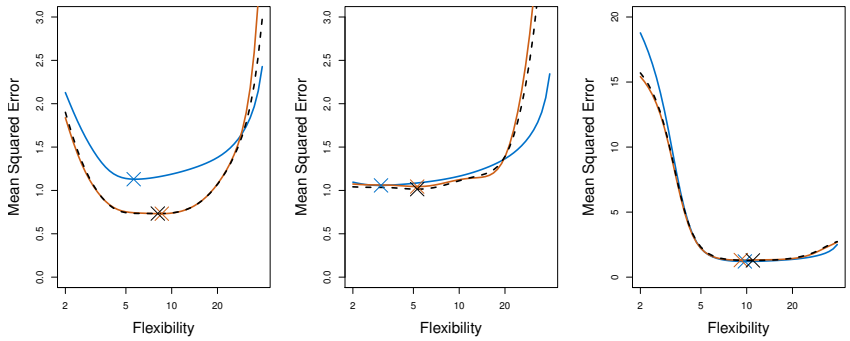
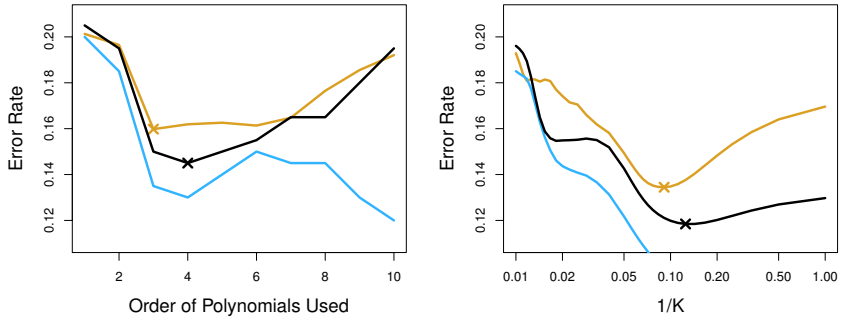


Figure 5.8 is for the simulation settings in Figures 2.13.



- 5.1.4: Bias–variance trade-off in choice of k

  - As k decreases, bias increases (because of overestimation of test error rate due to smaller sample size) but variance decreases (because the k models are less similar to each other).
  - When n is large (say, $n \geq 10000$), 5-fold CV is often enough.
  - When n is not very large, 10-fold CV (to avoid bias).
  - When n is small (say, $n \leq 200$), may repeat 10-fold CV several times to reduce the noise introduced by random partitioning. An example is in my week 2 notes.
- 5.1.5: An example of using CV on classification.

Bias in validation: For model assessment, validation and CV can overestimate test error rate when n is small. (Reason: In CV, all models are trained on datasets of size n_{train}, and thus we effectively evaluate the prediction performance for a model that is *built on a dataset of sample size* n_{train}. What we are really interested in is the prediction performance for the final model that is *built on all available data*, which has size n. Since $n_{train} < n$, CV has a tendency to overestimate the test error rate, and the smaller sample size the more variability in a fitted model. When $n = 1000$ and $n_{train} = 800$, the amount of overestimation is probably ignorable. When $n = 100$ and $n_{train} = 80$, the amount of overestimation can be non-trivial.)

For model selection: This bias may not have a large impact on model selection as long as the bias is similar across all models evaluated.

One-standard-error rule: Described near the end of ISLR 6.1. A nice illustration is from ESL Figure 3.7.

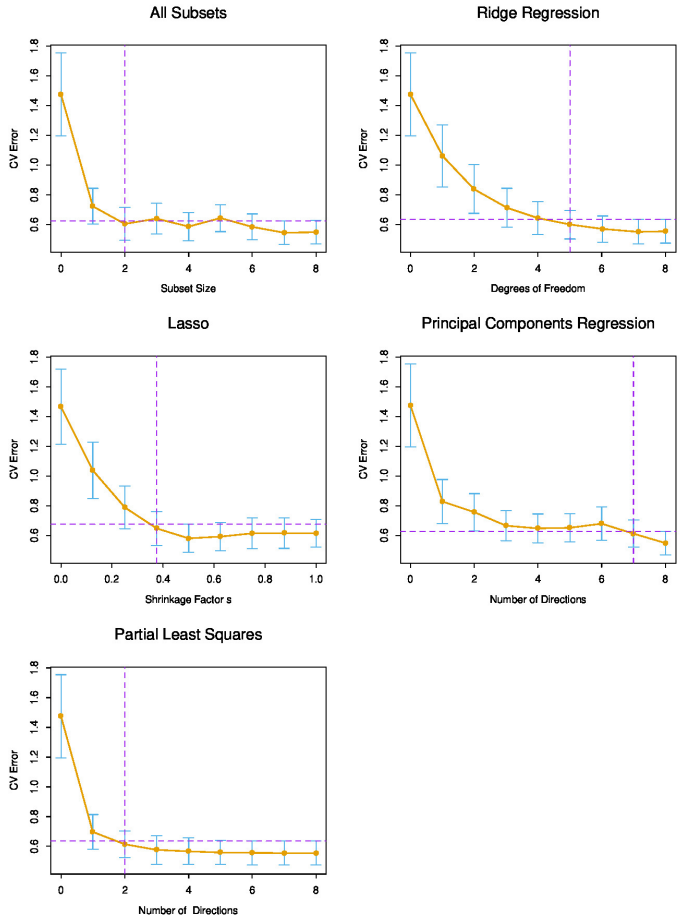


FIGURE 3.7. *Estimated prediction error curves and their standard errors for the various selection and shrinkage methods. Each curve is plotted as a function of the corresponding complexity parameter for that method. The horizontal axis has been chosen so that the model complexity increases as we move from left to right. The estimates of prediction error and their standard errors were obtained by tenfold cross-validation; full details are given in Section 7.10. The least complex model within one standard error of the best is chosen, indicated by the purple vertical broken lines.*

5.2 Simulation for the scenarios in ISLR 4.5

I wrote a function to generate a partition of the indices $\{1, 2, \cdots, n\}$ into K subsets.

```
cv.part = function(n, K) {
    s1 = n %/% K; s2 = n %% K
    pos2 = ifelse(rep(s2==0,K), (1:K)*s1, (1:K)*s1 + c(rep(0,K-s2), 1:s2))
    pos1 = c(1, pos2[-K]+1)
    part = sample(n,n)
    out=list()
    for(k in 1:K) out[[k]] = part[pos1[k]:pos2[k]]
    out
}
```

In `cv.part()`, if n can be divided by K, the subsets have equal size; if not, some of the subsets have an extra index. In `cv.glm()` in the R `boot` package, `sample(rep(1:K, ceiling(n/K)), n)` is used to generate a partition, which can lead to an uneven distribution. For example, when $n = 7$, $K = 3$, it can give a partition with subset sizes (1,3,3), while (2,2,3) is more desirable; when $n = 9$, $K = 4$, it can give a partition with subset sizes (0,3,3,3), while (2,2,2,3) is more desirable. The above function `cv.part()` always give the most desirable subset size distribution.

I wrote another function to perform K-fold CV for kNN in a range of k. The function returns `Nmiss` (the number of misclassification for every k in the range) and `k` (the k that has the smallest Nmiss; if the smallest Nmiss is achieved by multiple k's, report the largest such k).

```
cv.knn = function(train, class, krange, K) {
  ## train: X data frame
  ## class: Y vector
  ## krange: a vector of k values to be evaluated for k-NN
  ## K: K-fold CV to be used

  require(class)  ## for knn()
  n = length(class)
  part = cv.part(n, K)

  Nmiss = NULL
  for(k in krange) {
    nmiss = 0
    for(ii in 1:K) {
      idxtest = part[[ii]]
      nmiss = nmiss + sum(class[idxtest] !=
                          knn(train[-idxtest, ], train[idxtest, ], class[-idxtest], k))
    }
  Nmiss[k] = nmiss
  }
  names(Nmiss) = krange

  kminmiss = max(krange[which(Nmiss == min(Nmiss))])
  list(Nmiss=Nmiss, k=kminmiss)
}
```

I try to simulate Scenario 4 in ISLR 4.5. Unfortunately I do not know all the details the authors used. You can experiment other values and other scenarios.

```
library(MASS)  ## for mvrnorm() and LDA/QDA
library(class)  ## for knn()
set.seed(2019)

ntrain=50; ntest=1000
mu1=c(0,0);  Sigma1=matrix(c(1,.5,.5,1),2)
```

```
mu2=c(1,1); Sigma2=matrix(c(1,-.5,-.5,1),2)

## testing data of size ntest
t1 = mvrnorm(ntest, mu=mu1, Sigma=Sigma1)
t2 = mvrnorm(ntest, mu=mu2, Sigma=Sigma2)
ttx = data.frame(rbind(t1,t2))
tty = rep(c(0,1), each=ntest)

## simulate 100 datasets each of size ntrain, apply the 5 methods
ntally = matrix(,100,5)  ## 5 methods to be evaluated
for(ii in 1:100) {
  d1 = mvrnorm(ntrain, mu=mu1, Sigma=Sigma1)
  d2 = mvrnorm(ntrain, mu=mu2, Sigma=Sigma2)
  dd = data.frame(y=rep(c(0,1), each=ntrain), rbind(d1,d2))
  ntally[ii, ] = c(
    sum(tty != knn(dd[,2:3], ttx, dd$y, k=1)),  ## 1-NN
    sum(tty != knn(dd[,2:3], ttx, dd$y, k=cv.knn(dd[,2:3], dd$y, k=1:20, 10)$k)),  ## kNN-CV
    sum(tty != predict(lda(y ~ X1+X2, data = dd), ttx)$class),  ## LDA
    sum(tty != (predict(glm(y ~ X1+X2, data=dd, family=binomial), ttx, type='response') >0.5)),
    sum(tty != predict(qda(y ~ X1+X2, data = dd), ttx)$class))  ## QDA
}

methodnames = c('1nn','knn-CV','LDA','Logistic','QDA')
plot(factor(rep(factor(methodnames, levels=methodnames), each=100)), as.numeric(ntally))
```

5.3 ISLR 5.2 Bootstrap

Motivation: If we knew the true underlying distribution for our data, we would be able to evaluate a measure of interest, e.g., the variance of a complicated statistic calculated from a dataset of size n. We would do it by repeatedly sampling datasets of size n from the underlying distribution, calculating the statistic of interest for each simulated dataset, and then computing the variance of the statistic.

We can mimic this by treating the empirical distribution obtained from data as if it were the underlying distribution. Assuming no ties, the empirical distribution is multinomial with n categories each having $1/n$ probability. To sample a dataset of size n from this multinomial distribution is effectively to **sample with replacement** n data points from the original dataset. This is also true when there are ties.

0.632: The probability for an observation to be sampled into a bootstrap sample. (The correct value is $1-(1-\frac{1}{n})^n$, which is approximately $1-e^{-1}=0.632$.)

For example, we have data from an unknown distribution and we can estimate the 75th percentile of the distribution. We want to know the accuracy of this estimate. The `boot` package has a function `boot()` for this. It requires a function with two arguments: `data`, `index`.

```
library(boot)  ## for boot() and cv.glm()

## generate data from a mixture distribution
x = c(rnorm(40,0,2), runif(30,0,5), rgamma(30, 2, .3))
hist(x)
```

Now we obtain the bootstrap distribution for sample variance.

```
myvar = function(data, index) var(data[index])
myvar(x)

aa = boot(x, myvar, R=100)
aa; names(aa)
```

```
hist(aa$t)
aa$t0; mean(aa$t)-aa$t0; sd(aa$t)
```

Now we obtain the bootstrap distribution for 75th percentile.

```
my75th = function(data, index) quantile(data[index], .75)
my75th(x)

aa = boot(x, my75th, R=100)
aa

hist(aa$t)
aa$t0; mean(aa$t)-aa$t0; sd(aa$t)
```

5.4 Assignment

1. **Homework**: ISLR Chapter 5 Exercises 9 (`Boston` dataset in MASS)