Stats216v: Statistical Learning

Stanford University
Summer 2017
Gyu-Ho Lee (gyuhox@gmail.com (mailto:gyuhox@gmail.com))

5. Resampling Methods Â¶

5.1.R

When we fit a model to data, which is typically larger?

1. Test Error
2. Training Error

Gyu-Ho's Answer: 1.

Training error almost always underestimates test error, sometimes dramatically

5.1.R2

What are reasons why test error could be LESS than training error?

1. By chance, the test set has easier cases than the training set.
2. The model is highly complex, so training error systematically overestimates test error.
3. The model is not very complex, so training error systematically overestimates test error.

Gyu-Ho's Answer: 1, 3.

1.

Training error usually UNDERestimates test error when the model is very complex (compared to the training set size), and is a pretty good estimate when the model is not very complex. However, it's always possible we just get too few hard-to-predict points in the test set, or too many in the training set.

5.2.R1

Suppose we want to use cross-validation to estimate the error of the following procedure:

Step 1: Find the k variables most correlated with y

Step 2: Fit a linear regression using those variables as predictors

We will estimate the error for each k from 1 to p, and then choose the best k.

True or false: a correct cross-validation procedure will possibly choose a different set of k variables for every fold.

Gyu-Ho's Answer: True.

True: we need to replicate our entire procedure for each training/validation split. That means the decision about which k variables are the best must be made on the basis of the training set alone. In general, different training sets will disagree on which are the best k variables.

chapter-05-resampling-1.png

chapter-05-resampling-2.png

chapter-05-resampling-3.png

So I've got my samples here and my predictors here. And now in this first approach, we first select the best set of predictors based on the correlation with the outcome, that's over here. And we keep these predictors and throw the rest away. And now in step two, we're going to apply cross-validation. What does that mean? We divide the data up into, say, five parts. We apply our classifier to four parts.

chapter-05-resampling-4.png

Key point being, though, that we form the folds before we filter or fit to the data. So that we're applying cross-validation to the entire process, not just the second step. So that's the right way to do it.

That with a large number of predictors, if you filter them you've got to include that in your cross-validation. And if you don't, you can incur a serious error in your cross-validation estimate.

chapter-05-resampling-5.png

5.3.R1

Suppose that we perform forward stepwise regression and use cross-validation to choose the best model size.

Using the full data set to choose the sequence of models is the WRONG way to do cross-validation (we need to redo the model selection step within each training fold). If we do cross-validation the WRONG way, which of the following is true?

1. The selected model will probably be too complex.
2. The selected model will probably be too simple.

Gyu-Ho's Answer: 1.

Using the full data set to choose the best variables means that we do not pay as much price as we should for overfitting (since we are fitting to the test and training set simultaneously). This will lead us to underestimate test error for every model size, but the bias is worst for the most complex models. Therefore, we are likely to choose a model that is more complex than the optimal model.

5.4.R1

One way of carrying out the bootstrap is to average equally over all possible bootstrap samples from the original data set (where two bootstrap data sets are different if they have the same data points but in different order). Unlike the usual implementation of the bootstrap, this method has the advantage of not introducing extra noise due to resampling randomly. (You can use "^" to denote power, as in "n^2")

To carry out this implementation on a data set with n data points, how many bootstrap data sets would we need to average over?

Gyu-Ho's Answer: n^n .

Completely removing the bootstrap resampling noise is usually not worth incurring the extreme computational cost. If B is large, but still less than n^n, random resampling gives a good Monte Carlo estimate of the idealized bootstrap estimate for all n^n data sets.

chapter-05-bootstrap-01.png

chapter-05-bootstrap-02.png

chapter-05-bootstrap-03.png

The test error is the average error that results from using a statistical learning method to predict the response on a new observation, one that was not used in training the method. In contrast, the training error can be easily calculated by applying the statistical learning method to the observations used in its training. But the training error rate often is quite different from the test error rate, and in particular the former can dramatically underestimate the latter.

You should be aware of the fact that prediction error estimates are subject to bias and variance. With small sample sizes, the variance is the problem. To get a first idea of the variance of your prediction error estimates, look at the variance of prediction error within and between bootstrapped models that are built using the same input variables. While this can underestimate the true variance, it will give you a first idea of what is going on.

chapter-05-bootstrap-04.png

chapter-05-bootstrap-05.png

chapter-05-bootstrap-06.png

chapter-05-bootstrap-07.png

chapter-05-bootstrap-08.png

chapter-05-bootstrap-09.png

chapter-05-bootstrap-10.png

chapter-05-bootstrap-11.png

chapter-05-bootstrap-12.png

5.5.R

If we have n data points, what is the probability that a given data point does not appear in a bootstrap sample?

Gyu-Ho's Answer: $(1 - \frac{1}{n})^n$.

To construct a bootstrap sample, we repeatedly draw a single data point from a sample of size n, n times. Any given data point has a $1 - \frac{1}{n}$ chance of not being selected in each draw. Hence, the chance of not being selected in any of the n draws is $(1 - \frac{1}{n})^n$

5.Q.1

If we use ten-fold cross-validation as a means of model selection, the cross-validation estimate of test error is:

1. biased upward
2. biased downward
3. unbiased
4. potentially any of the above

Gyu-Ho's Answer: 4.

There are competing biases: on one hand, the cross-validated estimate is based on models trained on smaller training sets than the full model, which means we will tend to overestimate test error for the full model.

On the other hand, cross-validation gives a noisy estimate of test error for each candidate model, and we select the model with the best estimate. This means we are more likely to choose a model whose estimate is smaller than its true test error rate, hence, we may underestimate test error. In any given case, either source of bias may dominate the other.

5.Q.2

Why can't we use the standard bootstrap for some time series data?

1. The data points in most time series aren't i.i.d. (independent and identically distributed).
2. Some points will be used twice in the same sample.
3. The standard bootstrap doesn't accurately mimic the real-world data-generating mechanism.

Gyu-Ho's Answer: 1, 3.

The bootstrap always involves using some points more than once in each resample, but that doesn't inherently make it incorrect (unless we are trying to gauge prediction error). The real problem in this case is that the usual bootstrap algorithm samples i.i.d., so there is no serial autocorrelation (unlike what is observed in most time series). This makes the set of resampled time series very very different from the sorts of time series we actually get in the real world.

5.5.R1

Download the file 5.R.RData and load it into R using `load("5.R.RData")`. Consider the linear regression model of y on $X1$ and $X2$. What is the standard error for β_1 ?

In [4]:
```
loaded = load("5.R.RData")
head(loaded)

rData = get(loaded)
head(rData)

ls() # 'Xy' 'loaded' 'rData'
```

'Xy'

X1	X2	y
1.297720	0.8059212	0.2989683
1.267323	0.7990341	0.3181337
1.236882	0.7921693	0.3372015
1.206317	0.7852963	0.3561210
1.175553	0.7783848	0.3748415
1.144513	0.7714042	0.3933122

'Xy' 'loaded' 'rData'

In [6]:
```
# Use summary(lm(y~.,data=Xy))
model.lm = lm(y~X1+X2, data=Xy)
summary(model.lm)
```

```
Call:
lm(formula = y ~ X1 + X2, data = Xy)

Residuals:
     Min       1Q   Median       3Q      Max 
-1.44171 -0.25468 -0.01736  0.33081  1.45860 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  0.26583    0.01988  13.372  < 2e-16 ***
X1           0.14533    0.02593   5.604 2.71e-08 ***
X2           0.31337    0.02923  10.722  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.5451 on 997 degrees of freedom
Multiple R-squared:  0.1171,    Adjusted R-squared:  0.1154 
F-statistic: 66.14 on 2 and 997 DF,  p-value: < 2.2e-16
```

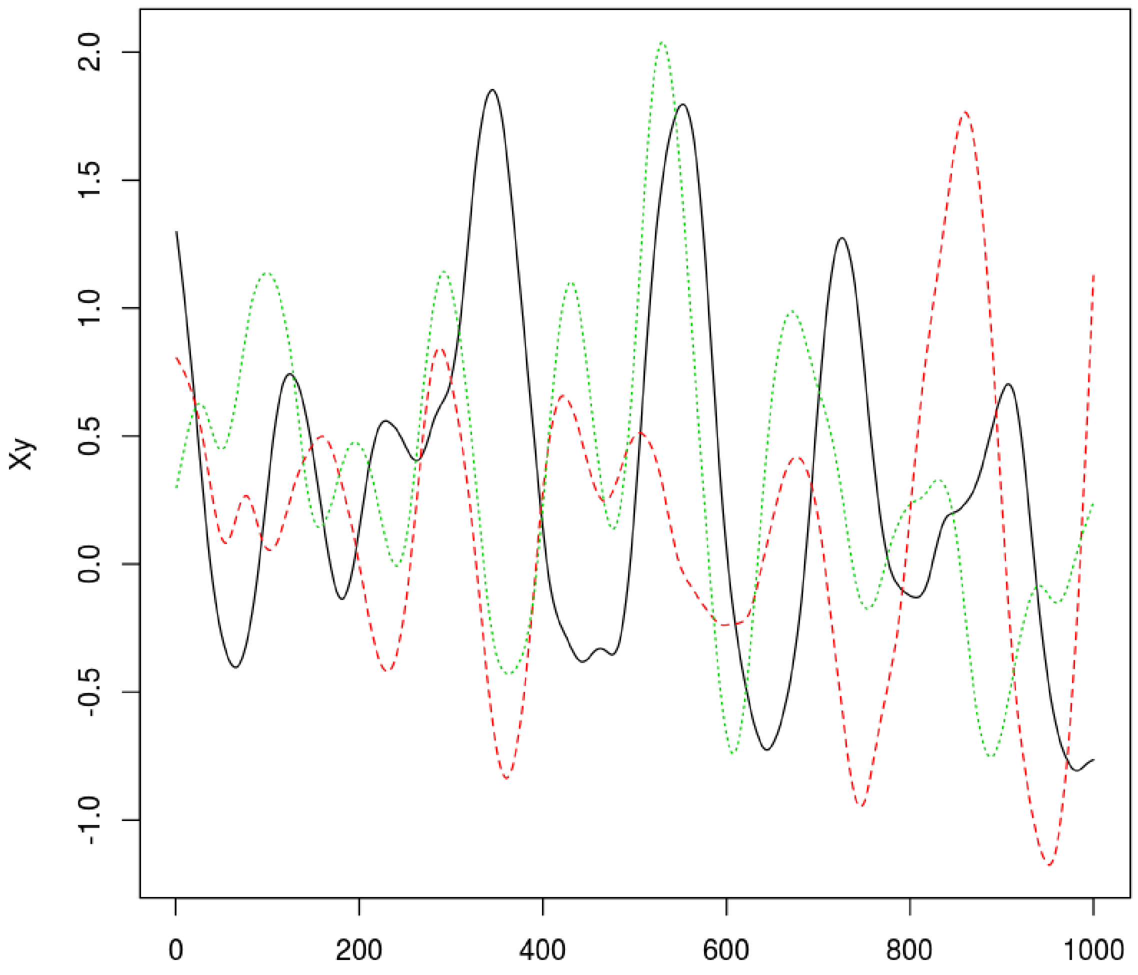
Gyu-Ho's Answer: 0.02593.

5.5.R2

Next, plot the data using $matplot(Xy, type = " \ l \ ")$. Which of the following do you think is most likely given what you see?

1. Our estimate of $s.e.(\hat{\beta}_1)$ is too high.
2. Our estimate of $s.e.(\hat{\beta}_1)$ is too low.
3. Our estimate of $s.e.(\hat{\beta}_1)$ is about right.

In [10]:
```
# Plot the columns of one matrix against the columns of another.
matplot(Xy, type="l")
```



Gyu-Ho's Answer: Our estimate of $s.e.(\hat{\beta}_1)$ is too low.

There is very strong autocorrelation between consecutive rows of the data matrix. Roughly speaking, we have about 10-20 repeats of every data point, so the sample size is in effect much smaller than the number of rows (1000 in this case).

5.R.R3

Now, use the (standard) bootstrap to estimate $s.e.(\hat{\beta}_1)$. To within 10%, what do you get?

```
In [43]: library(boot)

         alpha = function(x,y){
           vx = var(x)
           vy = var(y)
           cxy = cov(x,y)
           return ((vy-cxy)/(vx+vy-2*cxy))
         }

         alpha(Xy$X1, Xy$y)
```

```
0.416719162834766
```

```
In [44]: alpha.fn1 = function(data, index) {
           return (with(data[index,], alpha(Xy$X1, Xy$y)))
         }
```

```
In [45]: alpha.fn2 = function(data, index) {
           fitl = lm(y~., data=Xy[index,])
           return (coefficients(fitl)[['X1']])
         }
```

```
In [46]: set.seed(1)
         alpha.fn1(Xy, sample(1:100, 100, replace=TRUE))

         set.seed(1)
         alpha.fn2(Xy, sample(1:100, 100, replace=TRUE))
```

```
0.416719162834766

0.105906759887081
```

```
In [47]: boot.out = boot(Xy, alpha.fn2, R=1000)
         boot.out
```

```
ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = Xy, statistic = alpha.fn2, R = 1000)


Bootstrap Statistics :
     original       bias    std. error
t1* 0.1453263 0.0001885914  0.02873965
```

Gyu-Ho's Answer: 0.02873965.

When we do the i.i.d. bootstrap, we are relying on the original sampling having been i.i.d. That is the same assumption that screwed us up when we used lm.

5.R.R4

Finally, use the block bootstrap to estimate $s.e.(\hat{\beta}_1)$. Use blocks of 100 contiguous observations, and resample ten whole blocks with replacement then paste them together to construct each bootstrap time series. For example, one of your bootstrap resamples could be:

```
new.rows = c(101:200, 401:500, 101:200, 901:1000, 301:400, 1:100, 1:100,
801:900, 201:300, 701:800)
new.Xy = Xy[new.rows, ]
```

To within 10%, what do you get?

```
In [88]: boot.fn.ts = function(data){
           fit = lm(y~., data)
           return (coef(fit))
         }

         bootcorr = tsboot(Xy, boot.fn.ts, R=1000, sim="fixed", l=100)
         summary(bootcorr)
```

```
          Length Class      Mode
t0           3   -none-     numeric
t         3000   -none-     numeric
R            1   -none-     numeric
data         3   data.frame list
seed       626   -none-     numeric
statistic    1   -none-     function
sim          1   -none-     character
n.sim        1   -none-     numeric
call         6   -none-     call
l            1   -none-     numeric
endcorr      1   -none-     logical
```

In [91]:
```
sd(bootcorr$t)
```

0.228620657712559

0.2.

The block bootstrap does a better job of mimicking the original sampling procedure, because it preserves the autocorrelation.

In [76]:
```
library(boot)
load("5.R.RData", verbose=T)

bs = function(data, index){
  x = sample(seq(1, 901, 100), replace=T)
  indicies = as.vector(mapply(seq, from=x, to=x+99))
  return (summary(lm(y~., data=data[indicies,]))$coefficients[2,2])
}

boot(Xy, statistic=bs, R=10000)
```

```
Loading objects:
  Xy

ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = Xy, statistic = bs, R = 10000)


Bootstrap Statistics :
      original      bias    std. error
t1* 0.02036288 0.004409733 0.005342799
```