

Regression III

Lecture 8: Resampling Bootstrapping, Jackknifing, Cross-Validation

Dave Armstrong

University of Wisconsin – Milwaukee
Department of Political Science

e: armstrod@uwm.edu
w: www.quantoid.net/ICPSR.php

1 / 63

Goals of the Lecture

- Introduce the general idea of Resampling - techniques to resample from the original data
 - Bootstrapping
 - Jackknife
 - Cross-validation
- An extended example of bootstrapping local polynomial regression models.

2 / 63

Resampling: An Overview

- Resampling techniques sample from the original dataset
- Some of the applications of these methods are:
 - to compute standard errors and confidence intervals (either when we have small sample sizes, dubious distributional assumptions or for a statistic that does not have an easily derivable asymptotic standard error)
 - Subset selection in regression
 - Handling Missing Data
 - Selection of degrees of freedom in nonparametric regression (especially GAMs)
- For the most part, this lecture will discuss resampling techniques in the context of computing confidence intervals and hypothesis tests for regression analysis.

3 / 63

Resampling and Regression: A Caution

- There is no need whatsoever for bootstrapping in regression analysis if the OLS assumptions are met
 - In such cases, OLS estimates are unbiased and maximally efficient.
- There are situations, however, where we cannot satisfy the assumptions and thus other methods are more helpful
 - Robust regression (such as MM-estimation) often provides better estimates than OLS in the presence of influential cases, but only has reliable SEs asymptotically.
 - Local polynomial regression is often “better” (in the RSS sense) in the presence of non-linearity, but because of the unknown df, only has approximate sampling distribution.
- Cross-validation is particularly helpful for validating models and choosing model fitting parameters

4 / 63

Bootstrapping: General Overview

- If we assume that a random variable X or statistic has a particular population, we can study how a statistical estimator computed from samples behaves
- We don't always know, however, how a variable or statistic is distributed in the population
 - For example, there may be a statistic for which standard errors have not been formulated (e.g., imagine we wanted to test whether two additive scales have significantly different levels of internal consistency - Cronbach's α doesn't have an exact sampling distribution)
 - Another example is the impact of missing data on a distribution - we don't know how the missing data differ from the observed data
- Bootstrapping is a technique for estimating standard errors and confidence intervals (sets) without making assumptions about the distributions that give rise to the data

5 / 63

Bootstrapping: General Overview (2)

- Assume that we have a sample of size n for which we require more reliable standard errors for our estimates
 - Perhaps n is small, or alternatively, we have a statistic for which there is no known sampling distribution
- The bootstrap provides one "solution"
 - Take several new samples from the original sample, calculating the statistic each time
 - Calculate the average and standard error (and maybe quantiles) from the empirical distribution of the bootstrap samples
 - In other words, we find a standard error based on sampling (with replacement) from the original data
- We apply principles of inference similar to those employed when sampling from the population
 - The population is to the sample as the sample is to the bootstrap samples

6 / 63

Bootstrapping: General Overview (3)

there are several Variants of the bootstrap:

1. Nonparametric Bootstrap
 - No underlying population distribution is assumed
 - Most commonly used method
2. Smoothed bootstrap
 - Smooths the sample distribution and then samples from it
3. Parametric Bootstrap
 - Equivalent to Maximum Likelihood
 - Assumes that the statistic has a particular parametric form (e.g., normal)
4. Bayesian Bootstrap
 - Rather than give equal probability of being selected, assigns different probabilities to each case based on some prior knowledge

7 / 63

Jackknifing

- Jackknife also resamples the data
- It differs from bootstrapping in that rather than take several new random samples of the data with replacement, the jackknife procedure resamples the data by randomly taking a single observation out
 - In other words, new estimates of the statistics are calculated with a different observation deleted each time
- Unless the sample is very large, the number of jackknife samples used is usually equal to the number of cases in the original sample
- The average of the new estimate is then calculated to find the jackknife estimate
- Simulation studies generally show that the jackknife works well for robust estimators of location, but not as well as the bootstrap for estimating the standard deviation

8 / 63

Bootstrapping the Mean

- Imagine, unrealistically, that we are interested in finding the confidence interval for the mean of a sample of only 4 observations
- Specifically, assume that we are interested in the difference in income between husbands and wives
 - We have four cases, with the following mean differences (in \$ 1000's): 6, -3, 5, 3. for a mean of 2.75 and a standard deviation of 4.031
- From classical theory, we can calculate the confidence interval:

$$\begin{aligned}\mu &= \bar{Y} \pm t_{n-1, 0.025} \frac{S}{\sqrt{n}} \\ &= 2.75 \pm 4.30 \times \frac{4.31}{\sqrt{4}} \\ &= 2.75 \pm 8.66\end{aligned}$$

- Now we'll compare the confidence interval to the one calculated using bootstrapping

9 / 63

Defining the Random Variable

- The first thing that bootstrapping does is estimate the population distribution of Y from the four observations in the sample
- In other words, the random variable Y^* is defined:

Y^*	$p^*(Y^*)$
6	0.25
-3	0.25
5	0.25
3	0.25

- The mean of Y^* is then simply the mean of the sample:

$$\begin{aligned}E^*(Y^*) &= \sum Y^* p^*(Y^*) \\ &= 2.75 \\ &= \bar{Y}\end{aligned}$$

10 / 63

The Sample as the Population (1)

- We now treat the sample as if it were the population, and resample from it
- In this case, we take all possible samples with replacement, meaning that we take $n^n = 256$ different samples
- Since we found all possible samples, the mean of these samples is simply the original mean
- We then determine the standard error of \bar{Y} from these samples

$$SE^*(\bar{Y}) = \sqrt{\frac{\sum_{b=1}^{n^n} (\bar{Y}_b^* - \bar{Y})^2}{n^n}} = 1.745$$

We now adjust for the sample size

$$\hat{SE}(\bar{Y}) = \sqrt{\frac{n}{n-1}} SE^*(\bar{Y}^*) = \sqrt{\frac{4}{3}} \times 1.745 = 2.015$$

11 / 63

The Sample as the Population (2)

- In this example, because we used all possible resamples of our sample, the bootstrap standard error (2.015) is exactly the same as the original standard error
- This approach can be used for statistics for which we do not have standard error formulas, or we have small sample sizes
- In summary, the following analogies can be made to sampling from the population
 - Bootstrap observations → original observations
 - Bootstrap Mean → original sample mean
 - Original sample mean → unknown population mean μ
 - Distribution of the bootstrap means → unknown sampling distribution from the original sample

12 / 63

Characteristics of the Bootstrap Statistic

- The bootstrap sampling distribution around the original estimate of the statistic T is analogous to the sampling distribution of T around the population parameter θ
- The average of the bootstrapped statistics is simply:

$$\bar{T}^* = E(T^*) \approx \frac{\sum_{b=1}^R T_b^*}{R}$$

where R is the number of bootstraps

- The bias of T can be seen as its deviation from the bootstrap average (i.e., it estimates $T - \theta$)

$$\hat{B}^* = \bar{T}^* - T$$

- The estimated bootstrap variance of T^* is:

$$\hat{V}(T^*) = \frac{\sum_{b=1}^R (T_b^* - \bar{T}^*)^2}{R - 1}$$

13 / 63

Bootstrapping with Larger Samples

- The larger the sample, the more effort it is to calculate the bootstrap estimates
 - With large sample sizes, the possible number of bootstrap samples n^n gets very large and impractical (e.g., it would take a long time to calculate 1000^{1000} bootstrap samples)
 - typically we want to take somewhere between 1000 and 2000 bootstrap samples in order to find a confidence interval of a statistic
- After calculating the standard error, we can easily find the confidence interval. Three methods are commonly used
 1. Normal Theory Intervals
 2. Percentile Intervals
 3. Bias Corrected Percentile Intervals

14 / 63

Evaluating Confidence Intervals

Accuracy:

- how quickly do coverage errors go to zero?
- $\text{Prob}\{\theta < \hat{T}_{lo}\} = \alpha$ and $\text{Prob}\{\theta > \hat{T}_{up}\} = \alpha$
- Errors go to zero at a rate of:
 - $\frac{1}{n}$ (second-order accurate)
 - $\frac{1}{\sqrt{n}}$ (first-order accurate)

Transformation Respecting:

- For any monotone transformation of θ , $\phi = m(\theta)$, can we obtain the right confidence interval on $\hat{\phi}$ with the confidence intervals on $\hat{\theta}$ mapped by $m()$? E.g.,

$$[\hat{\phi}_{lo}, \hat{\phi}_{up}] = [m(\hat{\theta}_{lo}), m(\hat{\theta}_{up})]$$

15 / 63

Bootstrap Confidence Intervals: Normal Theory Intervals

- Many statistics are asymptotically normally distributed
- Therefore, in large samples, we may be able to use a normality assumption to characterize the bootstrap distribution. E.g.,

$$\hat{T}^* \sim N(\hat{T}, \hat{s}e^2)$$

where $\hat{s}e$ is $\sqrt{\hat{V}(T^*)}$

- This approach works well for the bootstrap confidence interval, but only if the bootstrap sampling distribution is approximately normally distributed
 - In other words, it is important to look at the distribution before relying on the normal theory interval

16 / 63

Bootstrap Confidence Intervals: Percentile Intervals

- Uses percentiles of the bootstrap sampling distribution to find the end-points of the confidence interval
- If $G\hat{G}$ is the CDF of T^* , then we can find the $100(1-\alpha)\%$ confidence interval with:

$$[\hat{T}_{\%,lo}, \hat{T}_{\%,up}] = [\hat{G}^{-1}(\alpha), \hat{G}^{-1}(1-\alpha)]$$

- The $(1 - 2\alpha)$ percentile interval can be approximated with:

$$[\hat{T}_{\%,lo}, \hat{T}_{\%,up}] \approx [T_B^{*(\alpha)}, T_B^{*(1-\alpha)}]$$

where $T_B^{*(\alpha)}$ and $T_B^{*(1-\alpha)}$ are the ordered (B) bootstrap replicates such that $100\alpha\%$ of them fall below the former and $100\alpha\%$ of them fall above the latter.

- These intervals do not assume a normal distribution, but they do not perform well unless we have a large original sample and at least 1000 bootstrap samples

17 / 63

Bootstrap Confidence Intervals: Bias-Corrected, Accelerated Percentile Intervals (BC_a)

- The BC_a CI adjusts the confidence intervals for bias due to small samples by employing a normalizing transformation through two correction factors.
- This is also a percentile interval, but the percentiles are not necessarily the ones you would think.
 - Using strict percentile intervals, $[\hat{T}_{lo}, \hat{T}_{up}] \approx [T_B^{*(\alpha)}, T_B^{*(1-\alpha)}]$
 - Here, $[\hat{T}_{lo}, \hat{T}_{up}] \approx [T_B^{*(\alpha_1)}, T_B^{*(\alpha_2)}]$
 - $\alpha_1 \neq \alpha$ and $\alpha_2 \neq (1 - \alpha_1)$

$$\alpha_1 = \Phi \left(\hat{z}_0 + \frac{\hat{z}_0 + z^{(\alpha)}}{1 - \hat{a}(\hat{Z}_0 + z^{(\alpha)})} \right)$$

$$\alpha_2 = \Phi \left(\hat{z}_0 + \frac{\hat{z}_0 + z^{(1-\alpha)}}{1 - \hat{a}(\hat{Z}_0 + z^{(1-\alpha)})} \right)$$

18 / 63

Bias correction: \hat{z}_0

$$\hat{z}_0 = \Phi^{-1} \left(\frac{\# T_b^* \leq T}{B} \right)$$

- This just gives the inverse of the normal CDF for proportion of bootstrap replicates less than T .
- Note that if $\# T_b^* \leq T$, $\hat{z}_0 = 0$.
- If T is unbiased, the proportion will be close to 0.5, meaning that the correction is close to 0.

19 / 63

Acceleration: \hat{a}

$$\hat{a} = \frac{\sum_{i=1}^n (T_{(\cdot)} - T_{(i)})^3}{6 \sum_{i=1}^n (T_{(\cdot)} - T_{(i)})^2}$$

- $T_{(i)}$ is the calculation of the original estimate T with each observation i jackknifed out in turn.
- $T_{(\cdot)}$ is $\sum_{i=1}^n \frac{T_{(i)}}{n}$
- The acceleration constant corrects for the fact that $se(T)$ is not the same for all true parameters θ as normal theory would suggest.

20 / 63

Evaluation Confidence Intervals

	Normal	Percentile	BC_a
Accuracy	1 st order	1 st order	2 nd order
Transformation-respecting	No	Yes	Yes

21 / 63

Bootstrapping the Median

The median doesn't have an exact sampling distribution (though there is a normal approximation). Let's consider bootstrapping the median to get a sense of its sampling variability.

```
> options(useFancyQuotes=F)
> set.seed(123)
> n <- 25
> x <- rchisq(n,1,1)
> meds <- rep(NA, 1000)
> for(i in 1:1000){
+   meds[i] <- median(x[sample(1:n, n, replace=T)])
+ }
> median(x)

[1] 1.206853

> summary(meds)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.2838  0.6693  1.2070  1.2900  1.7850  3.7400
```

22 / 63

Make CIs

- Normal Theory CI:

```
> se <- sd(meds)
> bias <- mean(meds) - median(x)
> norm.ci <- median(x) - bias + qnorm((1+.95)/2)*se*c(-1,1)
```

- Percentile CI:

```
> med.ord <- meds[order(meds)]
> pct.ci <- med.ord[c(25,975)]
```

- BC_a CI:

```
> zhat <- qnorm(mean(meds < median(x)))
> medi <- sapply(1:n, function(i)median(x[-i]))
> Tdot <- mean(medi)
> ahat <- sum(((Tdot-medi)^3)/(6*sum((Tdot-medi)^2)^(3/2)))
> zalpha <- 1.96
> z1alpha <- -1.96
> a1 <- pnorm(zhat + ((zhat + zalpha)/(1-ahat*(zhat + zalpha))))
> a2 <- pnorm(zhat + ((zhat + z1alpha)/(1-ahat*(zhat + z1alpha))))
> a1 <- floor(a1*1000)
> a2 <- ceiling(a2*1000)
> bca.ci <- med.ord[c(a2, a1)]
```

23 / 63

Looking at CIs

```
> mat <- rbind(norm.ci, pct.ci, bca.ci)
> rownames(mat) <- c("norm", "pct", "bca")
> colnames(mat) <- c("lower", "upper")
> round(mat, 3)
```

```
      lower upper
norm -0.046 2.294
pct   0.473 2.829
bca   0.473 2.211
```

24 / 63

With boot package

```
> library(boot)
> set.seed(123)
> med.fun <- function(dat, inds){
+   assign(".inds", inds, envir=.GlobalEnv)
+   med <- median(x[.inds])
+   remove(".inds", envir=.GlobalEnv)
+   med
+ }
> boot.med <- boot(x, med.fun, R=1000)
> boot.ci(boot.med)
```

BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1000 bootstrap replicates

CALL :
boot.ci(boot.out = boot.med)

Intervals :
Level Normal Basic
95% (-0.107, 2.331) (-0.415, 1.940)

Level Percentile BCa
95% (0.473, 2.829) (0.473, 2.211)
Calculations and Intervals on Original Scale

25 / 63

BS Difference in Correlations

We can bootstrap the difference in correlations as well.

```
> set.seed(123)
> library(car)
> data(Mroz)
> Mroz <- Mroz[order(Mroz$hc), ]
> cor.fun <- function(dat, inds, strata=Mroz$hc){
+   assign(".inds", inds, envir=.GlobalEnv)
+   tmp <- dat[.inds, ]
+   cor1 <- cor(tmp$age[1:458], tmp$lwg[1:458],
+   use="complete")
+   cor2 <- cor(tmp$age[459:753], tmp$lwg[459:753],
+   use="complete")
+   remove(".inds", envir=.GlobalEnv)
+   cor1-cor2
+ }
> boot.cor <- boot(Mroz, cor.fun, R=2000)
```

26 / 63

Bootstrap Confidence Intervals

```
> library(boot)
> boot.ci(boot.cor)
```

BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 2000 bootstrap replicates

CALL :
boot.ci(boot.out = boot.cor)

Intervals :
Level Normal Basic
95% (-0.3465, -0.0539) (-0.3485, -0.0549)

Level Percentile BCa
95% (-0.1442, 0.1494) (-0.2236, -0.0504)
Calculations and Intervals on Original Scale
Warning : BCa Intervals used Extreme Quantiles
Some BCa intervals may be unstable

27 / 63

Bootstrapping Regression Models

There are two ways to do this

- Random-X Bootstrapping
 - The regressors are treated as random
 - Thus, we select bootstrap samples directly from the observations and calculate the statistic for each bootstrap sample
- Fixed-X Bootstrapping
 - The regressors are treated as fixed - implies that the regression model fit to the data is "correct"
 - The fitted values of Y are then the expectation of the bootstrap
 - We attach a random error (usually resampled from the residuals) to each \hat{Y} which produces the fixed-x bootstrap sample Y_b^*
 - To obtain bootstrap replications of the coefficients, we regress Y_b^* on the fixed model matrix for each bootstrap sample

28 / 63

Bootstrapping Regression: Example (1)

- Recall the inequality example from earlier in the course.
- We fit a linear model of `secpay` on the interaction of `gini` and `democrat`.

```
> dat <- read.table("~/Desktop/ICPSR_Slides/Lecture 1/Weakliem.txt", header=T)
> dat <- dat[-c(25,49), ]
> mod <- lm(secpay ~ gini*democrat, data=dat)
> summary(mod)
```

```
Call:
lm(formula = secpay ~ gini * democrat, data = dat)
```

```
Residuals:
    Min       1Q   Median       3Q      Max
-0.179825 -0.046637 -0.004133  0.047674  0.155604
```

```
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.940766   0.059679   15.764 < 2e-16 ***
gini         0.004995   0.001516    3.294  0.00198 **
democrat     0.486071   0.088182    5.512  1.86e-06 ***
gini:democrat -0.010840  0.002476   -4.378  7.53e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Residual standard error: 0.07118 on 43 degrees of freedom
Multiple R-squared: 0.5176, Adjusted R-squared: 0.484
```

29 / 63

Fixed-X Bootstrapping

- If we are relatively certain that the model is the “right” model (i.e., it is properly specified), then this makes sense.

```
> set.seed(123)
> resid <- mod$residuals
> yhat <- mod$fitted
> reg.fun <- function(dat, inds){
+   assign(".inds", inds, envir=.GlobalEnv)
+   boot.e <- resid[.inds]
+   boot.y <- yhat + boot.e
+   mod <- lm(boot.y ~ gini*democrat, data=dat)
+   remove(".inds", envir=.GlobalEnv)
+   mod$coef
+ }
> boot.reg1 <- boot(dat, reg.fun, R=2000)
```

30 / 63

Fixed-X CIs

```
> sapply(1:4, function(x)boot.ci(boot.reg1,
+ type=c("perc", "bca"), index = x)$percent)[4:5,]
```

```
      [,1]      [,2]      [,3]      [,4]
[1,] 0.8297759 0.002145095 0.3186109 -0.015595518
[2,] 1.0512193 0.007850081 0.6541840 -0.006146045
```

```
> sapply(1:4, function(x)boot.ci(boot.reg1,
+ type=c("perc", "bca"), index = x)$bca)[4:5,]
```

```
      [,1]      [,2]      [,3]      [,4]
[1,] 0.8282077 0.002244871 0.3140740 -0.01550417
[2,] 1.0499282 0.007894346 0.6462295 -0.00608761
```

31 / 63

Random-X resampling (1)

- Random-X resampling is also referred to as observation resampling
- It selects B bootstrap samples (i.e., resamples) of the observations, fits the regression for each one, and determines the standard errors from the bootstrap distribution
- This is done fairly simply in **R** using the `boot` function from the `boot` library, too.

32 / 63

Random-X Resampling (2)

```
> set.seed(123)
> reg.fun2 <- function(dat, inds){
+   assign(".inds", inds, envir=.GlobalEnv)
+   tmp <- dat[.inds, ]
+   mod <- lm(secpay ~ gini*democrat, data=tmp)
+   remove(".inds", envir=.GlobalEnv)
+   mod$coef
+ }
> boot.reg2 <- boot(dat, reg.fun2, R=2000)
```

33 / 63

Random-X Resampling (3)

```
> sapply(1:4, function(x)boot.ci(boot.reg2,
+ type=c("perc", "bca"), index = x)$percent)[4:5,]
      [,1]      [,2]      [,3]      [,4]
[1,] 0.8498018 0.001561269 0.2795144 -0.016084204
[2,] 1.0559103 0.007746922 0.6674366 -0.004973372

> sapply(1:4, function(x)boot.ci(boot.reg2,
+ type=c("perc", "bca"), index = x)$bca)[4:5,]
      [,1]      [,2]      [,3]      [,4]
[1,] 0.8316138 0.002207272 0.2819084 -0.016289688
[2,] 1.0363747 0.008185233 0.6693275 -0.005186623
```

34 / 63

What if we Retained the Outliers?

```
> sapply(1:4, function(x)boot.ci(boot.reg1,
+ type=c("perc", "bca"), index = x)$percent)[4:5,]
      [,1]      [,2]      [,3]      [,4]
[1,] 1.033167 -0.005032171 -0.04789152 -0.012624739
[2,] 1.371368 0.003821731 0.49534057 0.003285463

> sapply(1:4, function(x)boot.ci(boot.reg1,
+ type=c("perc", "bca"), index = x)$bca)[4:5,]
      [,1]      [,2]      [,3]      [,4]
[1,] 1.043736 -0.004539726 -0.06545286 -0.012421852
[2,] 1.390705 0.004463468 0.47996050 0.003719149

> sapply(1:4, function(x)boot.ci(boot.reg2,
+ type=c("perc", "bca"), index = x)$percent)[4:5,]
      [,1]      [,2]      [,3]      [,4]
[1,] 0.8950708 -0.009549574 -0.1460839 -0.014192002
[2,] 1.5407004 0.006289619 0.5928488 0.004586769

> sapply(1:4, function(x)boot.ci(boot.reg2,
+ type=c("perc", "bca"), index = x)$bca)[4:5,]
      [,1]      [,2]      [,3]      [,4]
[1,] 0.921167 -0.011738526 -0.2010531 -0.013297540
[2,] 1.615511 0.005801091 0.5591056 0.005619287

>
```

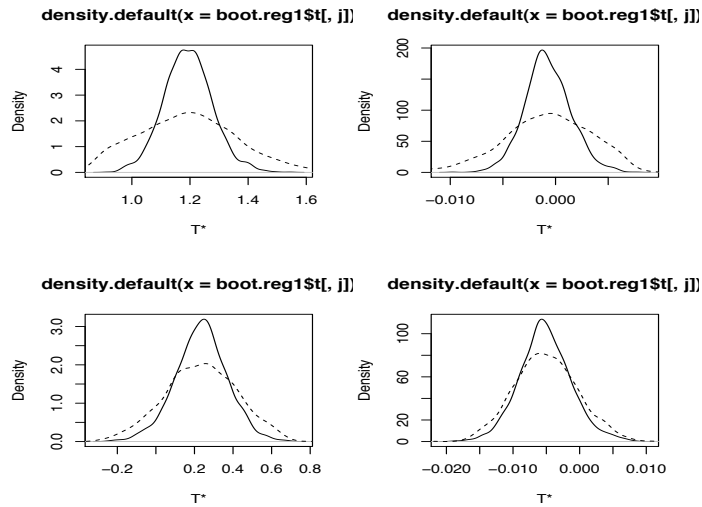
35 / 63

Bootstrap Distribution

```
> pdf("boot_dist.pdf", height=6, width=6)
> par(mfrow=c(2,2))
> for(j in 1:4){
+   plot(density(boot.reg1$t[,j]), xlab="T*", ylab="Density")
+   lines(density(boot.reg2$t[,j]), lty=2)
+ }
> invisible(dev.off())
```

36 / 63

The Bootstrap Distribution



37 / 63

Jackknife-after-Bootstrap

- The jackknife-after-bootstrap provides a diagnostic of the bootstrap by allowing us to examine what would happen to the density if particular cases were deleted
 - Given that we know that there are outliers, this diagnostic is important
 - The jackknife-after-bootstrap plot is produced using the `jack.after.boot` function in the `boot` package. Once again, the `index=2` argument asks for the results for the slope coefficient in the model
- ```
jack.after.boot(boot.reg1, index=2)
```

38 / 63

## Jackknife-after-Bootstrap (2)

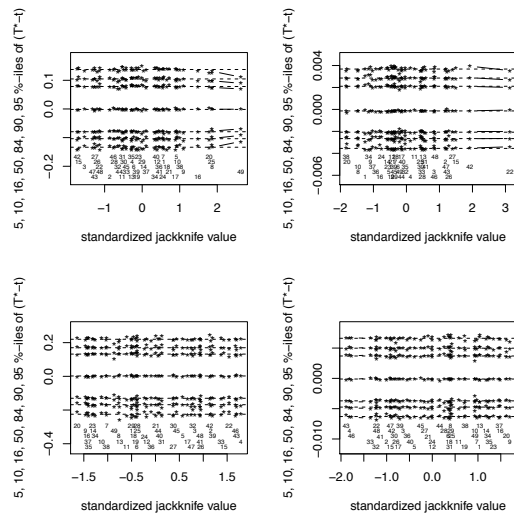
- The `jack.after.boot` plot is constructed as follows:
  - the horizontal axis represents the standardized jackknife value (the standardized value of the difference with that observation taken out)
  - The vertical axis represents various quantiles of the bootstrap statistic
    - Horizontal lines in the graph represent the bootstrap distribution at the various quantiles
  - Case numbers are labeled at the bottom of the graph so that each observation can be identified

39 / 63

## R code for Jack After Boot

```
> pdf("jab1.pdf", height=6, width=6)
> par(mfrow=c(2,2))
> jack.after.boot(boot.reg1, index=1)
> jack.after.boot(boot.reg1, index=2)
> jack.after.boot(boot.reg1, index=3)
> jack.after.boot(boot.reg1, index=4)
> invisible(dev.off())
> pdf("jab2.pdf", height=6, width=6)
> par(mfrow=c(2,2))
> jack.after.boot(boot.reg2, index=1)
> jack.after.boot(boot.reg2, index=2)
> jack.after.boot(boot.reg2, index=3)
> jack.after.boot(boot.reg2, index=4)
> invisible(dev.off())
```

40 / 63



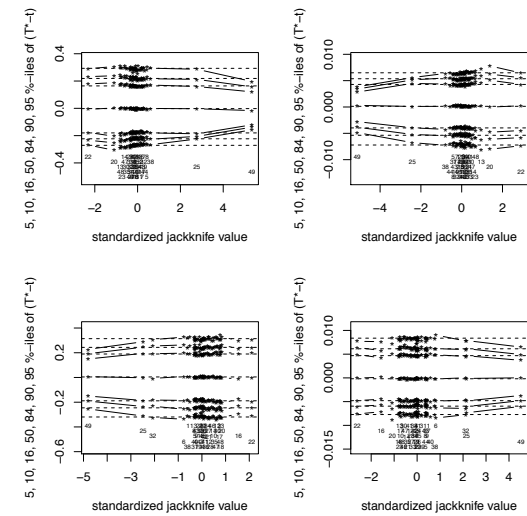
41 / 63

## Fixed versus random

Why do the samples differ?

- Fixed-X resampling enforces the assumption that errors are randomly distributed by resampling the residuals from a common distribution
- As a result, if the model is not specified correctly (i.e., there is un-modeled nonlinearity, heteroskedasticity or outliers) these attributes do *not* carry over to the bootstrap samples
  - the effects of outliers was clear in the random-X case, but not with the fixed-X bootstrap

43 / 63



42 / 63

## Bootstrapping the Loess Curve

We could also use bootstrapping to get the Loess Curve. First, let's try the fixed-x resampling:

```
> library(foreign)
> dat <- read.dta("~/Desktop/ICPSR_Slides/Lecture 6/jacob.dta")
> dat$perot <- (dat$perotvote - min(dat$perotvote))/
+ diff(range(dat$perotvote))
> dat$chal <- (dat$chal_vote - min(dat$chal_vote))/
+ diff(range(dat$chal_vote))
> seq.range <- function(x)seq(min(x), max(x), length=250)
> s <- seq.range(dat$perot)
> df <- data.frame(perot = s)
> lo.mod <- loess(chal ~ perot,
+ data=dat, span=.75)
> resid <- lo.mod$residuals
> yhat <- lo.mod$fitted
> lo.fun <- function(dat, inds){
+ assign(".inds", inds, envir=.GlobalEnv)
+ boot.e <- resid[.inds]
+ boot.y <- yhat + boot.e
+ tmp.lo <- loess(boot.y ~ perot,
+ data=dat, span=.75)
+ preds <- predict(tmp.lo, newdata=df)
+ remove(".inds", envir=.GlobalEnv)
+ preds
+ }
> boot.perot1 <- boot(dat, lo.fun, R=1000)
```

44 / 63

## Bootstrapping the Loess Curve (2)

### Now, random-X resampling

```
> lo.fun2 <- function(dat, inds){
+ assign(".inds", inds, envir=.GlobalEnv)
+ tmp.lo <- loess(boot.y ~ perot,
+ data=dat[,inds,], span=.75)
+ preds <- predict(tmp.lo, newdata=df)
+ remove(".inds", envir=.GlobalEnv)
+ preds
+ }
> boot.perot2 <- boot(dat, lo.fun, R=1000)
```

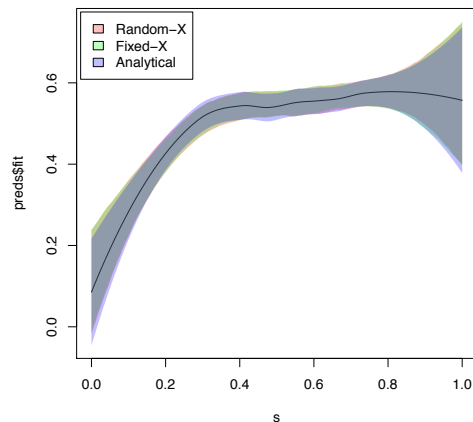
45 / 63

## Make Confidence Intervals

```
> out1 <- sapply(1:250, function(i)boot.ci(boot.perot1,
+ index=i, type="perc")$perc)
> out2 <- sapply(1:250, function(i)boot.ci(boot.perot2,
+ index=i, type="perc")$perc)
> preds <- predict(lo.mod, newdata=df, se=T)
> lower <- preds$fit - 1.96*preds$se
> upper <- preds$fit + 1.96*preds$se
> ci1 <- t(out1[4:5,])
> ci2 <- t(out2[4:5,])
> ci3 <- cbind(lower, upper)
> pdf("lo.ci.pdf", height=6, width=6)
> plot(s, preds$fit, type = "l",
+ ylim=range(c(ci1, ci2, ci3)))
> poly.x <- c(s, rev(s))
> polygon(poly.x, y=c(ci1[,1], rev(ci1[,2])),
+ col=rgb(1,0,0,.25), border=NA)
> polygon(poly.x, y=c(ci2[,1], rev(ci2[,2])),
+ col=rgb(0,1,0,.25), border=NA)
> polygon(poly.x, y=c(ci3[,1], rev(ci3[,2])),
+ col=rgb(0,0,1,.25), border=NA)
> legend("topleft",
+ c("Random-X", "Fixed-X", "Analytical"),
+ fill = rgb(c(1,0,0), c(0,1,0), c(0,0,1),
+ c(.25,.25,.25)), inset=.01)
> invisible(dev.off())
```

46 / 63

## Confidence Interval Graph



47 / 63

## Bootstrap Test of Non-linearity

We suggested before that the F-test when dealing with smoothers is only approximate for a couple of reasons:

- Distributional assumptions about the difference between  $\hat{f}(x)$  and  $f(x)$  may be unwarranted or unjustified.
- The degrees of freedom are only an approximation based on (various) analogies to the linear model.
- Yet to be discussed - when GCV and back-fitting are used to estimate the smoothing parameter on the penalized fit function, little is known about the distribution of the test statistic under  $H_0$ .

Bootstrapping can be a way of providing approximately the “right” answer in the face of these problems.

48 / 63

## Steps to Tests for Linearity

(from Keele's "Semiparametric Regression for the Social Sciences" p. 191).

1. First, estimate both the linear model and GAM such that:

$$\hat{y}_i = \gamma x_i + \beta Z$$
$$\tilde{y}_i = f(x_i) + \beta Z$$

2. Calculate  $t$ , the test statistic (here, the difference in model deviances).
3. Calculate the parametric model residuals  $\hat{e}_i = y_i - \hat{y}_i$ . And take bootstrap samples  $e_1^*, e_2^*, \dots, e_n^*$  from the original vector of residuals.
4. Calculate the bootstrap response:

$$y_i^* = \hat{y}_i + \hat{e}_i^*$$

and use the original predictors  $x_i$  and  $Z$  to estimate both the parametric and non-parametric models of  $y_i^*$  on  $x_i$  and  $Z$ . Use these results to re-calculate  $t^*$  and do this  $R$  times.

5. Calculate the new p-value:

$$\frac{[1 + \#(t^* \geq t)]}{(B + 1)}$$

49 / 63

## Test in R

```
> library(mgcv)
> library(boot)
> library(car)
> mod1 <- gam(prestige ~ log(income) + poly(women, 2) +
+ poly(education, 2), data=Prestige)
> mod2 <- gam(prestige ~ s(income, bs="cr") + poly(women, 2) +
+ poly(education, 2), data=Prestige)
> orig.t <- deviance(mod1) - deviance(mod2)
> resid <- mod1$residuals
> yhat <- mod1$fitted
> test.fun <- function(dat, inds){
+ assign(".inds", inds, envir=.GlobalEnv)
+ boot.e <- resid[.inds]
+ boot.y <- yhat + boot.e
+ tmp.mod1 <- gam(boot.y ~ log(income) +
+ poly(women, 2) + poly(education, 2),
+ data=dat)
+ tmp.mod2 <- gam(boot.y ~ s(income, bs="cr") +
+ poly(women, 2) + poly(education, 2),
+ data=dat)
+ remove(".inds", envir=.GlobalEnv)
+ deviance(tmp.mod1) - deviance(tmp.mod2)
+ }
```

50 / 63

## Looking at the Results

```
> boot.t <- boot(Prestige, test.fun, R=1000)
> (sum(boot.t$t >= orig.t)+1)/(1000+1)
[1] 0.2987013
> boot.ci(boot.t, type=c("perc", "bca"))
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1000 bootstrap replicates

CALL :
boot.ci(boot.out = boot.t, type = c("perc", "bca"))

Intervals :
Level Percentile BCa
95% (-104.5, 644.4) (6.5, 1004.9)
Calculations and Intervals on Original Scale
Some BCa intervals may be unstable
```

51 / 63

## When Might Bootstrapping Fail?

1. Incomplete Data: since we are trying to estimate a population distribution from the sample data, we must assume that the missing data are not problematic. It is acceptable, however, to use bootstrapping if multiple imputation is used beforehand
2. Dependent data: bootstrapping assumes independence when sampling with replacement. It should not be used if the data are dependent
3. Outliers and influential cases: if obvious outliers are found, they should be removed or corrected before performing the bootstrap (especially for random-X). We do not want the simulations to depend crucially on particular observations

52 / 63

### Cross-Validation (1)

- If no two observations have the same  $Y$ , a  $p$ -variable model fit to  $p + 1$  observations will fit the data precisely
- this implies, then, that discarding much of a dataset can lead to better fitting models
  - Of course, this will lead to biased estimators that are likely to give quite different predictions on another dataset (generated with the same DGP)
- Model validation allows us to assess whether the model is likely to predict accurately on future observations or observations not used to develop this model
- Three major factors that may lead to model failure are:
  1. Over-fitting
  2. Changes in measurement
  3. Changes in sampling
- External validation involves retesting the model on new data collected at a different point in time or from a different population
- Internal validation (or cross-validation) involves fitting and evaluating the model carefully using only one sample

53 / 63

### Cross-Validation (2)

- A basic, but powerful, tool for statistical model building
- Cross-validation is similar to bootstrapping in that it resamples from the original data
- The basic form involves randomly dividing the sample into two subsets:
  - The first subset of the data (screening sample) is used to select or estimate a statistical model
  - The second subset is then used to test the findings
- Can be helpful in avoiding capitalizing on chance and over-fitting the data - i.e., findings from the first subset may not always be confirmed by the second subsets
- Cross-validation is often extended to use several subsets (either a preset number chosen by the researcher or leave-one-out cross-validation)

54 / 63

### Cross-Validation (3)

- The data are split into  $k$  subsets (usually  $3 \leq k \leq 10$ ), but can also use leave-one-out cross-validation
- Each of the subsets are left out in turn, with the regression run on the remaining data
- Prediction error is then calculated as the sum of the squared errors:

$$RSS = \sum (Y_i - \hat{Y}_i)^2$$

- We choose the model with the smallest average "error"

$$MSE = \frac{\sum (Y_i - \hat{Y}_i)^2}{n}$$

- We could also look to the model with the largest average  $R^2$

55 / 63

### Cross-Validation (4)

How many observations should I leave out from each fit?

- There is no rule on how many cases to leave out, but Efron (1983) suggests that grouped cross-validation (with approximately 10% of the data left out each time) is better than leave-one-out cross-validation

Number of repetitions

- Harrell (2001:93) suggests that one may need to leave  $\frac{1}{10}$  of the sample out 200 times to get accurate estimates

Cross-validation does not validate the complete sample

- External validation, on the other hand, validates the model on a new sample
- Of course, limitations in resources usually prohibits external validation in a single study

56 / 63

## Cross-Validation in R

- Cross-validation is done easily using the `cv.glm` function in the `boot` package

```
> dat <- read.table("~/Desktop/ICPSR_Slides/Lecture 1/Weaklie")
> dat <- dat[-c(25,49),]
> mod1 <- glm(secpay ~ gini*democrat, data=dat)
> mod2 <- glm(secpay ~ gini+democrat, data=dat)
> cv.glm(dat, mod1, K=5)$delta
 1 1
0.006003098 0.005849918
> cv.glm(dat, mod2, K=5)$delta
 1 1
0.007896367 0.007757123
```

57 / 63

## Cross-validating the Prestige Model

```
> library(car)
> data(Prestige)
> mod1 <- glm(prestige ~ log(income) + women +
+ education, data=Prestige)
> mod2 <- glm(prestige ~ log(income) + poly(women,2) +
+ education, data=Prestige)
> mod3 <- glm(prestige ~ log(income) + poly(women,2) +
+ poly(education, 3), data=Prestige)
> cv.glm(Prestige, mod1, K=10)$delta
 1 1
52.18830 51.98022
> cv.glm(Prestige, mod2, K=10)$delta
 1 1
51.08067 50.79814
> cv.glm(Prestige, mod3, K=10)$delta
 1 1
50.78195 50.30153
```

58 / 63

## Cross-Validation with Splines

```
> dat <- read.dta("~/Desktop/ICPSR_Slides/Lecture 6/jacob.dta")
> dat$perot <- (dat$perotvote - min(dat$perotvote))/
+ diff(range(dat$perotvote))
> dat$chal <- (dat$chal_vote - min(dat$chal_vote))/
+ diff(range(dat$chal_vote))
> library(splines)
> mod1 <- glm(chal ~ perot, data=dat)
> mod2 <- glm(chal ~ poly(perot, 2), data=dat)
> mod3 <- glm(chal ~ poly(perot, 3), data=dat)
> mod4 <- glm(chal ~ bs(perot, df=4,
+ Boundary.knots=c(0,1)), data=dat)
> mod5 <- glm(chal ~ bs(perot, df=5,
+ Boundary.knots=c(0,1)), data=dat)
> mod6 <- glm(chal ~ bs(perot, df=6,
+ Boundary.knots=c(0,1)), data=dat)
> cv.glm(dat, mod1, K=10)$delta
 1 1
0.02818722 0.02816832
> cv.glm(dat, mod2, K=10)$delta
 1 1
0.02721040 0.02717115
> cv.glm(dat, mod3, K=10)$delta
 1 1
0.02680013 0.02674960
> cv.glm(dat, mod4, K=10)$delta
 1 1
0.02674414 0.02668138
> cv.glm(dat, mod5, K=10)$delta
 1 1
0.02704796 0.02696625
> cv.glm(dat, mod6, K=10)$delta
 1 1
0.02770366 0.02758329
```

59 / 63

## Cross-validating Span in Loess

We could use cross-validation to tell us something about the span in our Loess model.

1. First, split the sample into  $K$  groups (usually 10).
2. For each of the  $k = 10$  groups, estimate the model on the other 9 and get predictions for the omitted groups observations. Do this for each of the 10 subsets in turn.
3. Calculate the CV error:  $\frac{1}{n} \sum (y_i - \hat{y}_i)^2$
4. Potentially, do this lots of times and average across the CV error.

```
> library(foreign)
> dat <- read.dta("~/Desktop/ICPSR_slides/Lecture 6/jacob.dta")
> dat$perot <- (dat$perotvote - min(dat$perotvote))/
+ diff(range(dat$perotvote))
> dat$chal <- (dat$chal_vote - min(dat$chal_vote))/
+ diff(range(dat$chal_vote))
> lo.mod <- loess(chal ~ perot, data=dat, span=.75)
```

60 / 63

## CV Loess in R

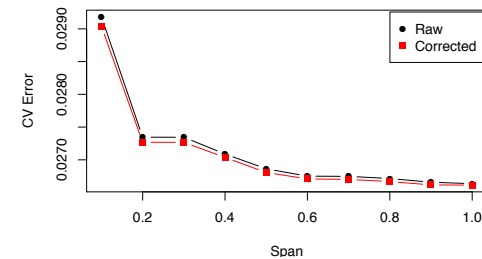
I took the code from the `cv.glm` function from the `boot` package and repurposed it for `loess`. This is in `cv.lo.R` which is on the course webpage.

```
> source("cv.lo.R")
> out.cv <- matrix(ncol=2, nrow=10)
> out.cv[1,] <- cv.lo(dat, "chal", update(lo.mod, span=.1),
+ numiter=25, K=10)
> out.cv[2,] <- cv.lo(dat, "chal", update(lo.mod, span=.2),
+ numiter=25, K=10)
> out.cv[3,] <- cv.lo(dat, "chal", update(lo.mod, span=.3),
+ numiter=25, K=10)
> out.cv[4,] <- cv.lo(dat, "chal", update(lo.mod, span=.4),
+ numiter=25, K=10)
> out.cv[5,] <- cv.lo(dat, "chal", update(lo.mod, span=.5),
+ numiter=25, K=10)
> out.cv[6,] <- cv.lo(dat, "chal", update(lo.mod, span=.6),
+ numiter=25, K=10)
> out.cv[7,] <- cv.lo(dat, "chal", update(lo.mod, span=.7),
+ numiter=25, K=10)
> out.cv[8,] <- cv.lo(dat, "chal", update(lo.mod, span=.8),
+ numiter=25, K=10)
> out.cv[9,] <- cv.lo(dat, "chal", update(lo.mod, span=.9),
+ numiter=25, K=10)
> out.cv[10,] <- cv.lo(dat, "chal", update(lo.mod, span=1),
+ numiter=25, K=10)
```

61 / 63

## Results

```
> s <- seq(.1,1, by=.1)
> pdf("cvres.pdf", height=4, width=6)
> plot(s, out.cv[,1], xlab = "Span", ylab="CV Error",
+ ylim=range(c(out.cv)), type="b", pch=16)
> lines(s, out.cv[,2], type="b", col="red", pch=15)
> legend("topright", c("Raw", "Corrected"),
+ pch=c(16,15), col=c("black", "red"),
+ inset=.01)
> invisible(dev.off())
```



62 / 63

## Summary and Conclusions

- resampling techniques are powerful tools for estimating standard errors from small samples or when the statistics we are using do not have easily determined standard errors
- Bootstrapping involves taking a “new” random sample (with replacement) from the original data
  - We then calculate bootstrap standard errors and statistical tests from the average of the statistic from the bootstrap samples
- Jackknife resampling takes new samples of the data by omitting each case individually and recalculating the statistic each time
- Cross-validation randomly splits the sample into two groups, comparing the model results from one sample to the results from the other

63 / 63