02:55 Wednesday 9th December, 2015
See updates and corrections at http://www.stat.cmu.edu/~cshalizi/mreg/

Lecture 28: The Bootstrap

36-401, Fall 2015, Section B

8 December 2015

Contents

1 Statistical Inference, Assuming Gaussian Noise

Consider our usual linear model

$$\mathbf{Y} = \mathbf{x}\beta + \epsilon$$

For a lot of results, it's enough to assume that

$$\mathbb{E}[\epsilon|\mathbf{x}] = \mathbf{0}$$

and

$$\mathrm{Var}\left[\epsilon|\mathbf{x}\right] = \sigma^2\mathbf{I}$$

These assumptions are enough to show the consistency of the least squares estimate

$$\widehat{\beta} = (\mathbf{x}^T\mathbf{x})^{-1}\mathbf{x}^T\mathbf{y}$$

The reason is that these assumptions let us write

$$\widehat{\beta} = \beta + (\mathbf{x}^T\mathbf{x})^{-1}\mathbf{x}^T\epsilon \ ,$$

i.e., they let us write the estimate as "true value plus weighted sum of the noise terms". Just as with simple linear regression, we can use this trick to get at a lot of properties of $\widehat{\beta}$: we can show that it's unbiased; that it has variance matrix $\sigma^2(\mathbf{x}^T\mathbf{x})^{-1}$; that, from the previous two properties, $\widehat{\beta} \to \beta$ as $n \to \infty$. But these assumptions are not enough to get useful hypothesis tests or confidence intervals.

For those, we have, so far, assumed that

$$\epsilon \sim N(\mathbf{0}, \sigma^2\mathbf{I})$$

independent of $\mathbf{x}$. This Gaussian noise assumption is important, because it gives us the distribution of $\widehat{\beta}$:

$$\widehat{\beta} \sim N(\beta, \sigma^2(\mathbf{x}^T\mathbf{x})^{-1})$$

And the reason for *that* is that if ϵ is Gaussian, then $(\mathbf{x}^T\mathbf{x})^{-1}\mathbf{x}^T\epsilon$ is a linear transformation of a Gaussian, which is also Gaussian. From knowing that $\widehat{\beta}$ is a Gaussian, everything we've done by way of statistical inference follows: the hypothesis tests, the confidence intervals, the prediction intervals, the F tests for multiple coefficients, etc. Without Gaussian noise, few of the formulas you memorized for the exam are, strictly, correct.

Looking at $Q-Q$ plots of our residuals is only important because we want them to be Gaussian. The only justification for every contemplating a Box-Cox transformation is that we'd like the noise to be Gaussian. We really have little reason to ever expect Gaussian noise; it's just very useful when it happens.

1.1 Other Parametric Distributions of the Noise

Suppose we didn't know think that ϵ was Gaussian, but still believed it was independently and identically distributed (IID). We might, for instance, think it had a "double exponential" (or "Laplacian") distribution, with probability density function

$$f(\epsilon) \propto e^{-|\epsilon|/L} \ ,$$

or was a scaled t distribution with a certain number ν of degrees of freedom,

$$f(\epsilon) \propto \left(1 + \frac{(x/s)^2}{\nu}\right)^{-(\nu+1)/2}$$

Both of these have heavier tails than the Gaussian distribution, so they can be very useful in practice.

If we think that the deterministic part of the model should still be linear, we still use least squares to get the estimate $\widehat{\beta}$, and it's still true that

$$\widehat{\beta} = \beta + (\mathbf{x}^T\mathbf{x})^{-1}\mathbf{x}^T\epsilon \tag{1}$$

But a linear combination of double-exponential variables is just a mess, as is a linear combination of t-distributions. A family of distributions where adding two random variables gives another distribution in the same family is called **stable**. The Gaussian distributions are stable, but most others aren't, and hence trying to work out an *exact* sampling theory for most other noise distributions, like the one we have for Gaussian noise, is pretty hopeless.

1.2 Asymptotic Gaussianity

Still, let's think about the noisy part of Eq. 1 some more. It's

$$(\mathbf{x}^T\mathbf{x})^{-1}\mathbf{x}^T\epsilon$$

Let's bundle up $(\mathbf{x}^T\mathbf{x})^{-1}\mathbf{x}^T$ as a matrix, call it $\mathbf{k}$. Then

$$\hat{\beta}_i = \beta_i + \sum_{j=1}^{n} k_{ij}\epsilon_j \tag{2}$$

If all of the k_{ij} were equal, we'd understand what was going on. After all, the central limit theorem says that when ϵ_j are IID with mean 0 and variance σ^2, their average tends towards a Gaussian distribution:

$$\sum_{j=1}^{n} \frac{1}{n}\epsilon_j \rightsquigarrow N(0, \sigma^2/n)$$

This is true whatever the distribution of the ϵ_j might be, provided, to repeat, that they're IID and they have mean 0 and variance $0 < \sigma^2 < \infty$. If instead of multiplying each ϵ_j by $1/n$ we multiplied them by some other constant, we'd change the variance but still tend towards a Gaussian:

$$\sum_{j=1}^{n} k\epsilon_j \rightsquigarrow N(0, \sigma^2 k^2 n)$$

On this basis, we might *hope* that, as $n \to \infty$,

$$\sum_{j=1}^{n} k_{ij}\epsilon_j \rightsquigarrow N(0, \sigma^2 \sum_j k_{ij}^2)$$

The difficulty is that the terms in the sum, while still statistically independent, are no longer *identically* distributed. There are central limit theorems which

apply to independent, non-identically distributed random variables, with the basic result being that if none of the k_{ij} is too big compared to the others, the sum is indeed asymptotically Gaussian[1].

If the matrix $\mathbf{x}$ doesn't give too much influence to any particular observations, then these central limit theorems usually apply, and we can say that

$$\widehat{\beta} \rightsquigarrow N(\beta, \sigma^2(\mathbf{x}^T\mathbf{x})^{-1}) \tag{3}$$

as $n \to \infty$, if ϵ is non-Gaussian but IID. From there, of course, all the usual formulas would also come to hold as $n \to \infty$.

How close to infinity does n have to be? You may have been told a bit of folklore which says that the central limit theorem dominates the behavior of a sample mean once $n > 30$. This is badly wrong even for averages, let alone more complicated functions like regression estimates. There is really no upper limit on how big n might have to be before Eq. 3 becomes a good approximation.

1.3 Summing Up on Gaussian Noise

To sum up, we have two situations:

1. We can assume that the noise is *exactly* Gaussian and independent, and have a nice body of theory for statistical inference, but we hardly ever see that happen.

2. We can assume that the noise is just independent, and recover the Gaussian theory asymptotically as $n \to \infty$, but we don't know what to do when n is finite, or even how big n has to get.

Clearly, this is an unsatisfying situation.

2 The Sampling Distribution as the Source of All Knowledge about Uncertainty

Our data comes from some distribution, let's say P. We would like to know some property of this distribution, say θ. (We may think of this as a regression coefficient, or the whole coefficient vector, or $\mathbb{E}[Y|X = x]$ for a particular x, etc.) Since we do not know P, we can't just calculate θ. What we can do, however, is draw a sample D from P, and then we calculate some statistic or other, $T(D)$. This serves as our estimate of θ. (The same goes for hypothesis tests, etc.) Because the data D are random, so is T. In fact, the distribution of T, the **sampling distribution** of our statistic, is set by the distribution of D. If we knew the sampling distribution, we'd know basically everything there is to know for statistical inference:

- The bias would be $\mathbb{E}[T] - \theta$

[1] If you want to follow this up, this is the "Lindeberg" central limit theorem.

- The standard error would $\sqrt{\mathrm{Var}\left[T\right]}$
- Hypothesis tests would come from quantiles of T
- Confidence intervals would come from inverting hypothesis tests

Unfortunately,

- Interesting statistics T are very complicated functions of the data, so their sampling distribution is complicated even if P is simple;
- Realistic distributions P are usually also complicated; and
- We don't know P anyway.

Put these together, and we should be surprised we can ever get nice, useful formulas for the sampling distribution of any statistic, rather than disappointed that we can't make it happen for regression without Gaussian noise.

The reason we can work out the sampling distributions for regression with Gaussian noise is that we (or rather, the ancestors) carefully adjusted the assumptions about the noise, the model, and the statistic *just so*, and everything came together. (E.g., we needed both a linear estimator and a noise distribution which was stable under linear combinations.) What could we do if we can't, or won't, fine-tune all the assumptions?

3 The Monte Carlo Principle

What is sometimes called the **Monte Carlo principle** is a general strategy for figuring out the behavior of complicated functions of complicated distributions: it says to simulate it and see what happens[2]. If we have samples $D_1, D_2, \ldots D_b$ from P, and we want to know the expectation of $T(D)$, we can approximate that as

$$\mathbb{E}\left[T(D)\right] \approx \frac{1}{b}\sum_{i=1}^{b} T(D_i) \equiv \overline{T}$$

If we want to know the variance, we can approximate that by

$$\mathrm{Var}\left[T(D)\right] \approx \frac{1}{b}\sum_{i=1}^{b} (T(D_i) - \overline{T})^2$$

If we want to know the q^{th} quantile of T, we can order the $T(D_i)$ from smallest to largest, and take the qb value as our estimate. If we want an interval which will

[2]The name, and to some extent the technique, originated with the physicists designing first the atomic and then the hydrogen bomb. Those designs required calculating the expectations of many elaborate functions of complex distributions (Serber, 1992). Rather than trying to actually do the integrals involved, they just developed efficient ways to sample from the distributions, and computed sample averages (Metropolis *et al.*, 1953). The spirit of "try it, and see what happens" went deep at Los Alamos...

contain T with probability $1-\alpha$, we order the $T(D_i)$ from smallest to largest, and exclude those with rank from 1 to $b\alpha/2$ on one side, and from $b(1-\alpha/2)$ to b on the other. And so on and so forth. All we need to do to get at any property of any function of the distribution P is to be able to draw from, or simulate, P.

One limitation of this for statistics is that, of course, we don't know the true P.

4 The Bootstrap Principle

The **bootstrap principle** is that if we have good approximation $\hat{P}$ to P, we can simulate from $\hat{P}$, and get a good approximation to the sampling distribution we want. That is, we apply the Monte Carlo principle to a distribution ($\hat{P}$) which we hope is close to the distribution we really care about (P).

More specifically, bootstrapping is always an algorithm, which goes, abstractly, as follows:

1. Observe data D, calculate estimate $T(D)$ and get an approximation $\hat{P}$ to P
2. Repeat b times:
   (a) Simulate surrogate data $\tilde{D}$ from $\hat{P}$.
   (b) Calculate $\tilde{T} = T(\tilde{D})$, just as those $\tilde{D}$ were real data
3. Approximate the distribution of T under P with the distribution of $\tilde{T}$ under $\hat{P}$

There are a *lot* of variants and refinements, some of which we will cover as this goes on, but in the meanwhile, we really need to be clearer about what "a good approximation $\hat{P}$ to P" might mean.

The two basic options are **model-based** bootstraps and **re-sampling** bootstraps[3]. In a model-based bootstrap, our approximation $\hat{P}$ is a full model of the data generating process, which we've estimated; we then simulate that model. In a re-sampling bootstrap, we treat the sample we observed as our best estimate at the distribution of the whole population, and so we draw a new sample from our original sample — we re-sample it. (If you like, in re-sampling the empirical distribution *is* our model.) Both of these ideas can be made a bit more concrete in the context of regression.

5 Bootstraps for Regression

Any regression model can be written as

$$Y = m(X) + \epsilon$$

[3]Often called these "parametric" and "non-parametric", respectively, but that's not quite as transparent, I think, as the other names.

with the caveat that the noise term ϵ might not have expectation zero[4] or be independent of X. Specifying the true regression function m and the distribution of the noise ϵ, including its dependence on X, gives us the data-generating distribution P.

Depending on what we are willing to believe about the true regression function m and the noise ϵ, we have different ways of coming up with approximations $\hat{P}$ to P, and different ways of simulating from those approximations.

5.1 The Linear, Gaussian Bootstrap

The simplest case we could have is where we think all of our usual modeling assumptions hold, so that $m(x) = x\beta$ and $\epsilon \sim N(0, \sigma^2)$, IID and independent of x. Then simulating from the estimated model is very easy.

```
# Simulate from a previously fitted linear model with Gaussian noise
# Inputs: model; data frame
# Outputs: new data frame with response values replaced
# Presumes: all necessary variables are in data frame
sim.lm.gauss <- function(mdl, df) {
    # What's the response variable called?
      # Should be the first variable in the vector of all variables
    resp.var <- all.vars(formula(mdl))[1]
    # What value should we expect for the response?
    expect.resp <- predict(mdl, newdata=df)
    # How big is the noise?
    sigma2.mle <- mean(residuals(mdl)^2)
    # Add appropriately-sized Gaussian noise to the response
    response <- expect.resp + rnorm(nrow(df),0,sqrt(sigma2.mle))
    df[,resp.var] <- response # Won't change df outside this function!
    return(df)
}
```

What we are doing in this function is creating a (very small!) imaginary or alternative world, the **simulation world** or **bootstrap world**, where we know that the model $Y = x\hat{\beta} + \epsilon$, $\epsilon \sim N(0, \hat{\sigma}^2)$ is exactly true. If $\hat{\beta} \approx \beta$ and $\hat{\sigma}^2 \approx \sigma^2$, and ϵ is Gaussian, then what we see in the simulation world is (close to) representative of what would happen in the real world if we could repeat our experiments many times. The advantage of the simulation world is that it's easy to re-run the simulation many times, whereas repeating the experiment may be expensive, difficult, unethical or flat-out impossible.

Of course, we don't just want to get a new data set, from an new simulation-world experiment; we want to know what we'd have concluded from that experiment. This is also easy.

```
# Re-estimate a linear model on a new data set
# Inputs: old model; data frame
```

[4] For instance, if the $m(X)$ is biased.

```
# Output: new lm object
# Presumes: data frame contains columns with appropriate names
re.lm <- function(mdl, df) {
    return(lm(formula(mdl), data=df))
}
```

Now if we want to get at the sampling distribution of, say, the estimated coefficient vector $\widehat{\beta}$, we just simulate it. We'll need *some* particular initial estimate to work with, so let's try to predict how much a cat's heart will weigh, from the cat's total body weight and its sex:

```
library(MASS); data(cats)
cats.lm <- lm(Hwt ~ Sex*Bwt, data=cats)
```

We now simulate from our `cats.lm` model many times (10000 is a conveniently small number), re-estimate the coefficients each time, and store the re-estimates in an array:

```
beta.boots <- replicate(10000,
                        coefficients(re.lm(cats.lm,
                                           sim.lm.gauss(cats.lm, cats))))
```

`beta.boots` is now a $4, 10000$ array, with one row for each coefficient, and 10000 columns, because we replicated the simulation 10000 times. Each column is a separate visit to the bootstrap world, where the true value of β is fixed to our initial estimate $\widehat{\beta}$. Symbolically, it looks like

$$\begin{bmatrix} \tilde{\beta}_{01} & \tilde{\beta}_{02} & \ldots & \tilde{\beta}_{0b} \\ \tilde{\beta}_{11} & \tilde{\beta}_{12} & \ldots & \tilde{\beta}_{1b} \\ \vdots & \vdots & \vdots & \vdots \\ \tilde{\beta}_{p1} & \tilde{\beta}_{p2} & \ldots & \tilde{\beta}_{pb} \end{bmatrix}$$

That being the case, we can get the bias:

```
rowMeans(beta.boots) - coefficients(cats.lm)
```

```
##  (Intercept)         SexM          Bwt     SexM:Bwt
## -0.004367952  0.005832001  0.002542342 -0.003549953
```

Each row contains all of our samples for one coefficient estimate, so the mean along each row is our (approximate) expected value of the estimate; we subtract the truth from that to get the bias.

We can also get the standard errors:

```
apply(beta.boots, 1, sd)
```

```
## (Intercept)        SexM         Bwt    SexM:Bwt
##   1.8048155   2.0222496   0.7596898   0.8205099
```

There is no `rowSDs` function, but the utility (or meta-) function `apply` lets us take any array (the first argument) and apply any function (the third argument) to either all its rows (middle argument `1`) or all its columns (middle argument `2`), or every entry in the array (middle argument `c(1,2)`). So the incantation above takes the standard deviation of each row.

To get a $1-\alpha$ confidence interval, we (conceptually) take all the values we got for each coefficient, sort them, and discard the lower and upper $\alpha/2$ tails. The appropriate incantation for 95% intervals is

```
apply(beta.boots, 1, quantile, prob=c(0.05/2, 1-0.05/2))
```

```
##       (Intercept)       SexM      Bwt   SexM:Bwt
## 2.5%   -0.4958377 -8.1368211 1.147514 0.07485683
## 97.5%   6.5214764 -0.1916686 4.103420 3.27873582
```

Now, none of this is actually *necessary* if we assume the truth is linear-and-Gaussian. We know that the bias is zero; we know that the standard deviations come from $\hat{\sigma}^2(\mathbf{x}^T\mathbf{x})^{-1}$; specifically, for the cats, they're

```
coefficients(summary(cats.lm))[,"Std. Error"]
```

```
## (Intercept)        SexM         Bwt    SexM:Bwt
##   1.8428394   2.0617552   0.7759022   0.8373255
```

We also know how to calculate the confidence intervals:

```
confint(cats.lm, level=0.95)
```

```
##                   2.5 %      97.5 %
## (Intercept) -0.6620801  6.62470490
## SexM        -8.2416012 -0.08919944
## Bwt          1.1024137  4.17041438
## SexM:Bwt     0.0208271  3.33170228
```

The fact that these numbers are *very* close to what we got by simulation tells us two things:

1. Simulation can work to replace intricate probabilistic mathematics with straightforward (even simple-minded) computations.
2. Simulation is completely redundant in the linear-Gaussian case, where we, and R, know all the formulas.

Simulation comes into its own when the formulas aren't available.

5.2 Linear, Non-Gaussian Noise

Suppose that we think the residuals follow some particular non-Gaussian distribution, which we know up to some set of parameters, e.g., a t distribution.

(We might have reached this conviction either because of some actual scientific theory, or by staring at the plot of the residuals.) If we know how to estimate the parameters of this noise distribution, and we can simulate from it, then we are in business.

I will illustrate this idea by using a t distribution for the noise in the model of cat's hearts. This is not an especially great model for this data, but it's only meant as an illustration[5]. We'll need to estimate the parameters of the t distribution; this job is already done for us by the function `fitdistr` in the `MASS` library.

```
# Simulate from a previously fitted linear model with t-distributed noise
# Inputs: model; data frame
# Outputs: new data frame with response values replaced
# Presumes: all necessary variables are in data frame
sim.lm.t <- function(mdl, df) {
    # What's the response variable called?
    resp.var <- all.vars(formula(mdl))[1]
    # What value should we expect for the response?
    expect.resp <- predict(mdl, newdata=df)
    # Estimate the t parameters, using MASS::fitdistr
    stopifnot(require(MASS)) # Make sure the library's available
    # After the example in help(fitdistr)
    mydt <- function(x, s, df) { dt(x/s, df)/s }
    t.params <- fitdistr(residuals(cats.lm), mydt, start=list(s=1, df=50),
                         lower=c(0,1))$estimate
    # Add appropriately-sized t-noise to the response
    response <- expect.resp + t.params["s"]*rt(nrow(df),df=t.params["df"])
    df[,resp.var] <- response # Won't change df outside this function
    return(df)
}
```

Having changed the function we use to simulate, absolutely nothing else needs to change:

```
beta.boots2 <- replicate(10000,
                         coefficients(re.lm(cats.lm,
                                            sim.lm.t(cats.lm, cats))))


apply(beta.boots2, 1, quantile, prob=c(0.05/2, 1-0.05/2))
```

```
##       (Intercept)       SexM      Bwt   SexM:Bwt
## 2.5%   -0.5914608 -8.1507654 1.129062 0.04982887
## 97.5%   6.5309158 -0.1609905 4.147760 3.29761116
```

[5] t-distributed noise is a much better idea in areas like finance.

5.3 Resampling Residuals

Suppose we are willing to believe that

$$Y = x\beta + \epsilon$$

and even that ϵ is independent of x, but not that we have any good idea about what the distribution of ϵ is. What are we to do?

Well, it will still be true that our residuals will give us an estimate of what the noise ϵ looks like — of what the true noise distribution is. We can use that. When we generate new data, we'll take $x\widehat{\beta} + \tilde{\epsilon}$, where $\tilde{\epsilon}$ is our simulated noise. Every time we need a value for $\tilde{\epsilon}$, we'll go to our vector of residuals and draw a random value from it — we will re-sample the residuals, with replacement. Here's how it works computationally:

```
# Simulate from a previously fitted linear model, resampling residuals
# Inputs: model; data frame
# Outputs: new data frame with response values replaced
# Presumes: all necessary variables are in data frame
sim.lm.residuals <- function(mdl, df) {
    # What's the response variable called?
    resp.var <- all.vars(formula(mdl))[1]
    # What value should we expect for the response?
    expect.resp <- predict(mdl, newdata=df)
    # Resample the residuals
    new.noise <- sample(residuals(mdl), size=length(expect.resp), replace=TRUE)
    # Add new noise to the expected response
    response <- expect.resp + new.noise
    df[,resp.var] <- response # Won't change df outside this function
    return(df)
}
```

When (as is usually the case) we want the re-sample to be exactly as large as the original sample, we can define a little convenience function:

```
resample <- function(x) { sample(x, size=length(x), replace=TRUE) }
```

(Note that, as written, this only works for vectors.) Let's see what kind of things this does to a short vector:

```
head(residuals(cats.lm))
```

```
##          1          2          3          4          5          6
## -1.2541405 -0.8541405  1.2458595 -1.3177819 -1.2177819 -0.9177819
```

```
resample(head(residuals(cats.lm)))
```

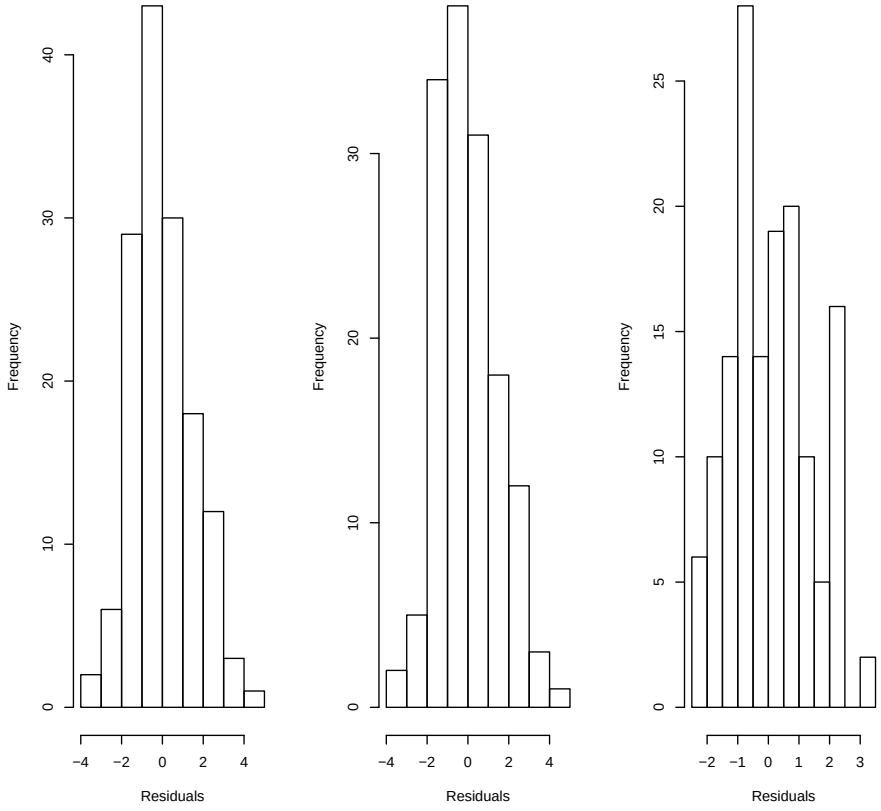
```
##          4          2          6          4          4          1
## -1.3177819 -0.8541405 -0.9177819 -1.3177819 -1.3177819 -1.2541405
```

```
resample(head(residuals(cats.lm)))
```

```
##          3          3          5          3          5          2
##  1.2458595  1.2458595 -1.2177819  1.2458595 -1.2177819 -0.8541405
```

Every time we run this function, we get different results; some samples get picked more than once, some don't get picked at all. When we look at the over-all distribution of each re-sample, it's somewhat less diverse than the sample we took it from, just as the sample is less diverse than the population, but it has the same over-all shape.

```
par(mfrow=c(1,3))
hist(residuals(cats.lm), main="", xlab="Residuals")
hist(resample(residuals(cats.lm)), main="", xlab="Residuals")
hist(resample(residuals(cats.lm)), main="", xlab="Residuals")
```



With this understood, once we have our simulator for resampling residuals, we have to make absolutely no changes to how we use it:

```
beta.boots3 <- replicate(10000,
                         coefficients(re.lm(cats.lm,
                                            sim.lm.residuals(cats.lm, cats))))
```

```
apply(beta.boots3, 1, quantile, prob=c(0.05/2, 1-0.05/2))
```

```
##       (Intercept)       SexM      Bwt   SexM:Bwt
## 2.5%   -0.6275111 -8.2046109 1.146750 0.04296844
## 97.5%   6.5361309 -0.1833857 4.162364 3.28889505
```

5.4 Resampling Cases

The last bootstrap we'll look at, here, is what's variously called the **case** (or **cases**) bootstrap, or the **pairs** bootstrap, or, more rarely, the **rows** bootstrap. The idea here is that our data consists of cases, each of which contains the predictor variables and the response variables, and we'll re-sample those whole points. The other names arise from thinking of each data point as a pair (x, y), or as a row in a data frame. We're using as our approximation $\hat{P}$ to the data-generating distribution P the joint empirical distribution of the predictors and the response.

The simulator is now sheer elegance in its simplicity:

```
# Re-sample the rows of a data frame
# Inputs: the data frame
# Output: a new data frame, contain a random sample, with replacement, of
  # rows from the input
resample.data.frame <- function(df) { df[resample(1:nrow(df)),] }
```

Notice that we don't use the estimated model here at all in the simulation — the procedure is quite agnostic as to whether our model is good or bad.

Having the simulator in hand, we can use it just like the others, and do inference using the simulation just like the others:

```
beta.boots4 <- replicate(10000,
                         coefficients(re.lm(cats.lm,
                                            resample.data.frame(cats))))
```

```
apply(beta.boots4, 1, quantile, prob=c(0.05/2, 1-0.05/2))
```

```
##       (Intercept)       SexM      Bwt SexM:Bwt
## 2.5%    0.1996264 -7.7712702 1.483763 0.273226
## 97.5%   5.7489468 -0.6295797 3.818110 3.103072
```

Resampling cases makes only very weak assumptions about the data-generating distribution, that all data points ((x, y) pairs) are independent and identically

distributed. It does not assume that any linear regression is correct[6], or that the noise is independent of x, or has constant variance.

6 Error in the Bootstrap, and Which Bootstrap When?

The bootstrap, remember, is a way of calculating the sampling distribution of T, under the true data-generating distribution P. There are two main sources of error in this calculation:

Simulation We'd like to see the full distribution of our statistic T under $\hat{P}$, but instead we only run b simulations under $\hat{P}$.

Approximation We're simulating from $\hat{P}$ rather than P.

Simulation error (or "Monte Carlo error") is easy to grasp, and in principle easy to control: the more simulations we run, the better. As $b \to \infty$, the simulation goes to zero. The only reason to ever restrict b is that each simulation does have some cost, in time if nothing else. In fact, for (most) inferential statistics, we can expect the simulation error to be $O(1/\sqrt{b})$, so if we keep increasing b we will experience *diminishing* returns, but never negative returns.

Approximation error comes from the fact that $\hat{P}$ is not P. This can itself be broken into two parts: estimation error (roughly, variance), and systematic distortion (roughly, bias). Estimation error arises because we only have a finite amount of data, say n observations, with which to estimate $\hat{P}$. Even if we resample cases, and so use the empirical distribution as our $\hat{P}$, we still only have n samples from the full population, which isn't all of it. Generally, estimation error will shrink to zero as $n \to \infty$, but it may shrink at different rates for different approximations. Systematic distortion is basically the approximation error which would be left even if we had infinite data — it comes from using a linear-Gaussian simulation when reality isn't linear or Gaussian, or resampling residuals when the noise is really heteroskedastic.

There is a trade-off when it comes to the two kinds of approximation error. The more we constrain $\hat{P}$ in advance of seeing any data, the stronger the assumption we put on it, the less we have to estimate, and so the smaller the estimation error. But, the true P doesn't obey those constraints, if our assumptions are wrong, the bigger the systematic distortion we're introducing. If our assumptions are right, using a more constrained $\hat{P}$ is pure advantage — basically, we're not wasting data figuring out that the constraints hold — but if those assumptions are wrong, they can easily make things worse.

Which bootstrap to use, then, depends on how strongly you trust your modeling assumptions.

- If you believe that the regression is linear and you know the distribution of the noise, use the fully model-based bootstraps.

[6] If the linear model *is* wrong, then we're doing statistical inference on the coefficients in the best linear approximation to the true regression function $m(x)$.

- If you believe that the regression is linear and the noise is independent of x, use resampling of residuals.
- If you are unwilling to believe that the noise is independent of x, and/or that the regression is truly linear, use resampling of cases.

(We'll cover the situation where you don't think the truth is linear but you are, somehow, convinced the noise is Gaussian when we go over fitting nonlinear models in 402.)

How do we tell? Well, in the first situation, all of the diagnostics we've been doing should look good, including the appropriate Q-Q plot. In the second situation, while the residuals should have the same distribution for all x, we don't care what that distribution is. Therefore when we plot residuals against predictors and fitted values, everything should look random, but not necessarily Gaussian, and the Q-Q plot need show no particular shape. In the third situation, by resampling cases we're still assuming independence across data points, so we should try to check that, but that's about all that we do need to check.

7 Further Reading

The bootstrap was introduced, by that name, by Efron (1979), in a remarkably accessible paper which is still worth reading. Related ideas, such as the "jackknife" (omit one data point, re-estimate, and look at the variance over all such re-estimates) go back at least to the 1940s, though they weren't systematically developed, or applied, until computing power got cheap enough to make something like the bootstrap feasible. A good systematic textbook is Davison and Hinkley (1997).

For the validity of case resampling even when all the usual linear-Gaussian assumptions fail, see, e.g., Buja *et al.* (2014). (That paper also shows how this bootstrap does the same job as the "robust standard errors" of econometrics.)

Next time, we will look at some of the techniques for reducing approximation error in bootstrap calculations, bootstrap prediction intervals, and what sorts of things the bootstrap can't do; all of these are also covered in Davison and Hinkley (1997).

References

Buja, Andreas, Richard Berk, Lawrence Brown, Edward George, Emil Pitkin, Mikhail Traskin, Linda Zhao and Kai Zhang (2014). "Models as Approximations: A Conspiracy of Random Regressors and Model Deviations Against Classical Inference in Regression." E-print, arxiv.org. URL `http://arxiv.org/abs/1404.1578`.

Davison, A. C. and D. V. Hinkley (1997). *Bootstrap Methods and their Applications*. Cambridge, England: Cambridge University Press.

Efron, Bradley (1979). "Bootstrap Methods: Another Look at the Jackknife." *Annals of Statistics*, **7**: 1–26. URL `http://projecteuclid.org/euclid.aos/1176344552`.

Metropolis, N., A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller and E. Teller (1953). "Equations of State Calculations by Fast Computing Machines." *Journal of Chemical Physics*, **21**: 1087–1092. doi:10.1063/1.1699114.

Serber, Robert (1992). *The Los Alamos Primer: The First Lectures on How to Build the Atomic Bomb*. Berkeley: University of California Press. Annotated by Robert Serber; edited and with an introduction by Richard Rhodes.