Bootstrapping Regression Models in R

An Appendix to *An R Companion to Applied Regression, Second Edition*

John Fox & Sanford Weisberg

last revision: 10 October 2017

Abstract

The *bootstrap* is a general approach to statistical inference based on building a sampling distribution for a statistic by resampling from the data at hand. This appendix to Fox and Weisberg (2011) briefly describes the rationale for the bootstrap and explains how to bootstrap regression models using the `Boot` function, which was added to the **car** package in 2012, and therefore is not described in Fox and Weisberg (2011). This function provides a simple way to access the power of the `boot` function (lower-case "b") in the **boot** package.

In 2017, the generic `Boot` function was extensively revised so that it now will work with many regression problems. Use of the function with the **betareg** and **crch** packages are given.

1 Basic Ideas

The *Bootstrap* is a general approach to statistical inference based on building a sampling distribution for a statistic by resampling from the data at hand. The term "bootstrapping," due to Efron (1979), is an allusion to the expression "pulling oneself up by one's bootstraps," in this case, using the sample data as a population from which repeated samples are drawn. At first blush, the approach seems circular, but has been shown to be sound.

At least two R packages for bootstrapping are associated with extensive treatments of the subject: Efron and Tibshirani's (1993) **bootstrap** package, and Davison and Hinkley's (1997) **boot** package. Of the two, **boot**, programmed by A. J. Canty, is somewhat more capable and is a part of the standard R distribution. The bootstrap is potentially very flexible and can be used in many different ways, and as a result using the **boot** package require some programming. In this appendix we will mostly discuss the **car** function `Boot`, which provides a simplified front-end to the **boot** package.

Confusion alert: `Boot` with a capital "B" is a function in the **car** package, and is the primary function used in the appendix. It is really just a convenience function that calls the function `boot` with a lower-case "b" in a package that is also called **boot**, also with a lower-case "b". We hope you will get a kick out of all the boots.

There are several forms of the bootstrap, and, additionally, several other resampling methods that are related to it, such as *jackknifing*, *cross-validation*, *randomization tests*, and *permutation tests*. We will stress the *nonparametric bootstrap*.

Suppose that we draw a sample $\mathbf{S} = \{X_1, X_2, ..., X_n\}$ from a population $\mathbf{P} = \{x_1, x_2, ..., x_N\}$; imagine further, at least for the time being, that N is very much larger than n, and that $\mathbf{S}$ is a simple random sample[1]. We will briefly consider other sampling schemes at the end of the appendix. It is helpful initially to think of the elements of the population and, hence, of the sample as scalar values, but they could just as easily be vectors.

Suppose that we are interested in some statistic $T = t(\mathbf{S})$ as an estimate of the corresponding population parameter $\theta = t(\mathbf{P})$. Again, θ could be a vector of parameters and T the corresponding vector of estimates, but for simplicity assume that θ is a scalar. A traditional approach to statistical inference is to make assumptions about the structure of the population, such as an assumption of normality, and, along with the stipulation of random sampling, to use these assumptions to derive the sampling distribution of T, on which classical inference is based. In certain instances, the exact distribution of T may be intractable, and so we instead derive its asymptotic distribution. This familiar approach has two potentially important deficiencies:

1. If the assumptions about the population are wrong, then the corresponding sampling distribution of the statistic may be seriously inaccurate. If asymptotic results are relied upon, these may not hold to the required level of accuracy in a relatively small sample.

2. The approach requires sufficient mathematical prowess to derive the sampling distribution of the statistic of interest. In some cases, such a derivation may be prohibitively difficult.

In contrast, the nonparametric bootstrap allows us to estimate the sampling distribution of a statistic empirically without making assumptions about the form of the population, and without deriving the sampling distribution explicitly. The essential idea of the nonparametric bootstrap is as follows: We proceed to draw a sample of size n from among the elements of the sample $\mathbf{S}$, sampling with replacement. Call the resulting *bootstrap sample* $\mathbf{S}_1^* = \{X_{11}^*, X_{12}^*, ..., X_{1n}^*\}$. It is necessary to sample with replacement, because we would otherwise simply reproduce the original sample $\mathbf{S}$. In effect, we are treating the sample $\mathbf{S}$ as an estimate of the population $\mathbf{P}$; that is, each element X_i of $\mathbf{S}$ is selected for the bootstrap sample with probability $1/n$, mimicking the original selection of the sample $\mathbf{S}$ from the population $\mathbf{P}$. We repeat this procedure a large number of times, R, selecting many bootstrap samples; the bth such bootstrap sample is denoted $\mathbf{S}_b^* = \{X_{b1}^*, X_{b2}^*, ..., X_{bn}^*\}$.

The key bootstrap analogy is therefore as follows:

The population is to the sample
as
the sample is to the bootstrap samples.

Next, we compute the statistic T for each of the bootstrap samples; that is $T_b^* = t(\mathbf{S}_b^*)$. Then the distribution of T_b^* around the original estimate T is analogous to the sampling distribution of the estimator T around the population parameter θ. For example, the average of the bootstrapped statistics,

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

[1]Alternatively, $\mathbf{P}$ could be an infinite population, specified, for example, by a probability distribution function.

estimates the expectation of the bootstrapped statistics; then $\widehat{B}^* = \overline{T}^* - T$ is an estimate of the bias of T, that is, $T - \theta$. Similarly, the estimated bootstrap variance of T^*,

$$\widehat{\text{Var}}^*(T^*) = \frac{\sum_{b=1}^{R}(T_b^* - \overline{T}^*)^2}{R-1}$$

estimates the sampling variance of T. The square root of this quantity

$$\widehat{\text{SE}}^*(T^*) = \sqrt{\frac{\sum_{b=1}^{R}(T_b^* - \overline{T}^*)^2}{R-1}}$$

is the bootstrap estimated standard error of T.

The random selection of bootstrap samples is not an essential aspect of the nonparametric bootstrap, and at least in principle we could enumerate *all* bootstrap samples of size n. Then we could calculate $E^*(T^*)$ and $\text{Var}^*(T^*)$ exactly, rather than having to estimate them. The number of bootstrap samples, however, is astronomically large unless n is tiny.[2] There are, therefore, two sources of error in bootstrap inference: (1) the error induced by using a particular sample $\mathbf{S}$ to represent the population; and (2) the sampling error produced by failing to enumerate all bootstrap samples. The latter source of error can be controlled by making the number of bootstrap replications R sufficiently large.

2 Bootstrap Confidence Intervals

There are several approaches to constructing bootstrap confidence intervals. The *normal-theory interval* assumes that the statistic T is normally distributed, which is often approximately the case for statistics in sufficiently large samples, and uses the bootstrap estimate of sampling variance, and perhaps of bias, to construct a $100(1-\alpha)\%$ confidence interval of the form

$$\theta = (T - \widehat{B}^*) \pm z_{1-\alpha/2}\widehat{\text{SE}}^*(T^*)$$

where $z_{1-\alpha/2}$ is the $1-\alpha/2$ quantile of the standard-normal distribution (e.g., 1.96 for a 95% confidence interval, when $\alpha = .05$).

An alternative approach, called the *bootstrap percentile interval*, is to use the empirical quantiles of T_b^* to form a confidence interval for θ:

$$T^*_{(\text{lower})} < \theta < T^*_{(\text{upper})}$$

where $T^*_{(1)}, T^*_{(2)}, \ldots, T^*_{(R)}$ are the ordered bootstrap replicates of the statistic; lower $= [(R+1)\alpha/2]$; upper $= [(R+1)(1-\alpha/2)]$; and the square brackets indicate rounding to the nearest integer. For example, if $\alpha = .05$, corresponding to a 95% confidence interval, and $R = 999$, then lower $= 25$ and upper $= 975$.

The *bias-corrected, accelerated* (or BC_a) *percentile intervals* perform somewhat better than the percentile intervals just described. To find the BC_a interval for θ:

- Calculate

$$z = \Phi^{-1}\left[\frac{\overset{R}{\underset{b=1}{\#}}(T_b^* \le T)}{R+1}\right]$$

[2]If we distinguish the order of elements in the bootstrap samples and treat all of the elements of the original sample as distinct (even when some have the same values) then there are n^n bootstrap samples, each occurring with probability $1/n^n$.

where $\Phi^{-1}(\cdot)$ is the standard-normal quantile function, and $\#\left(T_b^* \leq T\right)/(R+1)$ is the (adjusted) proportion of bootstrap replicates at or below the original-sample estimate T of θ. If the bootstrap sampling distribution is symmetric, and if T is unbiased, then this proportion will be close to .5, and the correction factor z will be close to 0.

- Let $T_{(-i)}$ represent the value of T produced when the ith observation is deleted from the sample;[3] there are n of these quantities. Let $\overline{T}$ represent the average of the $T_{(-i)}$; that is $\overline{T} = \sum_{i=1}^{n} T_{(-i)}/n$. Then calculate

$$a = \frac{\sum_{i=1}^{n}\left(\overline{T} - T_{(-i)}\right)^3}{6\left[\sum_{i=1}^{n}\left(T_{(-i)} - \overline{T}\right)^2\right]^{\frac{3}{2}}}$$

- With the correction factors z and a in hand, compute

$$\begin{aligned} a_1 &= \Phi\left[z + \frac{z - z_{1-\alpha/2}}{1 - a(z - z_{1-\alpha/2})}\right] \\ a_2 &= \Phi\left[z + \frac{z + z_{1-\alpha/2}}{1 - a(z + z_{1-\alpha/2})}\right] \end{aligned}$$

  where $\Phi(\cdot)$ is the standard-normal cumulative distribution function. The values a_1 and a_2 are used to locate the endpoints of the corrected percentile confidence interval:

$$T^*_{(\text{lower*})} < \theta < T^*_{(\text{upper*})}$$

  where lower* = $[Ra_1]$ and upper* = $[Ra_2]$. When the correction factors a and z are both 0, $a_1 = \Phi(-z_{1-\alpha/2}) = \Phi(z_{\alpha/2}) = \alpha/2$, and $a_2 = \Phi(z_{1-\alpha/2}) = 1 - \alpha/2$, which corresponds to the (uncorrected) percentile interval.

To obtain sufficiently accurate 95% bootstrap percentile or BC$_a$ confidence intervals, the number of bootstrap samples, R, should be on the order of 1000 or more; for normal-theory bootstrap intervals we can get away with a smaller value of R, say, on the order of 100 or more, because all we need to do is estimate the standard error of the statistic.

3 Bootstrapping Regressions

Recall Duncan's regression of prestige on income and education for 45 occupations from Chapters 1 and 6 in Fox and Weisberg (2011)[4]. In the on-line appendix on robust regression, we refit this regression using an M-estimator with the Huber weight function, employing the `rlm` function in the **MASS** package, which is available when you load the `car` package:

[3]The $T_{(-i)}$ are called the *jackknife values* of the statistic T. Although we will not pursue the subject here, the jackknife values can also be used as an alternative to the bootstrap to find a nonparametric confidence interval for θ.

[4]R functions used but not described in this appendix are discussed in Fox and Weisberg (2011) All the R code in this appendix can be downloaded from `http://tinyurl.com/carbook`. Alternatively, if you are running R and attached to the Internet, load the **car** package and enter the command `carWeb(script="appendix-bootstrap")` to view the R command file for the appendix in your browser.

```
library(car)
library(MASS)
mod.duncan.hub <- rlm(prestige ~ income + education, data=Duncan, maxit=200)
summary(mod.duncan.hub)

##
## Call: rlm(formula = prestige ~ income + education, data = Duncan, maxit = 200)
## Residuals:
##    Min     1Q Median     3Q    Max
## -30.12  -6.89   1.29   4.59  38.60
##
## Coefficients:
##             Value  Std. Error t value
## (Intercept) -7.111  3.881     -1.832
## income       0.701  0.109      6.452
## education    0.485  0.089      5.438
##
## Residual standard error: 9.89 on 42 degrees of freedom
```

The coefficient standard errors reported by `rlm` rely on asymptotic approximations, and may not be trustworthy in a sample of size 45. Let us turn, therefore, to the bootstrap. We set the `maxit` argument to `rlm` in anticipation of the bootstrap, because some of the bootstrap samples may need more iterations to converge.

There are two general ways to bootstrap a regression like this: We can treat the predictors as *random*, potentially changing from sample to sample, or as *fixed*. We will deal with each case in turn, and then compare the two approaches. For reasons that should become clear in the subsequent sections, random-x resampling is also called *case resampling*, and fixed-x resampling is also called *residual resampling*.

3.1 Random-x or Case Resampling

Broadening the scope of the discussion, assume that we want to fit a regression model with response variable y and predictors $x_1, x_2, \ldots, x_k$. We have a sample of n observations $\mathbf{z}'_i = (y_i, x_{i1}, x_{i2}, \ldots, x_{ik})$, $i = 1, \ldots, n$. In random-x or case resampling, we simply select R bootstrap samples of the $\mathbf{z}'_i$, fitting the model and saving the coefficients from each bootstrap sample. This is the default method used by the `Boot` function in `car`.

The call to the `Boot` function has up to five arguments:

```
Boot(object, f = coef, labels = names(f(object)), R = 999,
    method = c("case", "residual"), ...)
```

Only the first argument is required, and it must be the name of a regression object such as the object `mod.duncan.hub` we just created with the call to `rlm`. We will discuss in Section 5 the conditions that need to be satisfied for an object to work with the `Boot` function. The argument `f` is the name of a function that will be computed on each bootstrap replication. The default is the `coef` function, which for most regression objects will return the vector of regression coefficient estimates. If this argument is to set `f = coef` meaning that the coefficient estimates will be computed and saved on each replication. If you wanted a bootstrap distribution for the scale factor in an `rlm` fit, you could use `f = sigmaHat`, since the **car** function `sigmaHat` returns the scale factor. You could return both the coefficients and the scale factor with `f = function(mod){c(coef(mod), sigmaHat(mod))}`. The argument `labels` provides labels for the quantities that are kept on each iteration. If not set, the program selects labels. In the default case of `f = coef`, the program will use the coefficient labels but in other cases the labels used may not be very descriptive. For example, `labels=c(labels(coef(mod)), "sigmaHat")` would be appropriate if `f` returns both coefficient estimates and the scale estimate. The next argument, `method`, can be either `"case"`, the default for case resampling, or `"residual"` for residual resampling discussed later in this appendix. The final `...` argument permits passing additional arguments to the `boot` function.

For the example of Duncan's data fit with a Huber M estimate, we specify

```
set.seed(12345) # for reproducibility
system.time(duncan.boot <- Boot(mod.duncan.hub, R=1999))

## Loading required namespace:  boot

##    user  system elapsed
##    5.89    0.00    5.91
```

We ran `Boot` within a call to the function `system.time` to provide a sense of how long a bootstrapping operation like this takes. In this case, we generated $R = 1999$ bootstrap replicates of the regression coefficients. The first number returned by `system.time` is the CPU (processing) time for the operation, in seconds, while the third number is the total elapsed time. Here, both CPU and elapsed time are several seconds.[5] Although this is a small problem, the time spent depends more upon the number of bootstrap samples than upon the sample size. There are two ways to think about waiting several seconds for the bootstrapping to take place: On the one hand, we tend to be spoiled by the essentially instantaneous response that R usually provides, and by this standard several seconds seems a long time. On the other hand, bootstrapping is not an exploratory procedure, and a brief wait is a trivial proportion of the time typically spent on a statistical investigation.

The `Boot` function returns an object, here named `boot.duncan`, of class `"boot"`. The **car** package includes a `summary` method,

```
summary(duncan.boot, high.moments=TRUE)

##                R original bootBias bootSE bootMed bootSkew bootKurtosis
## (Intercept) 1999   -7.111  0.13965  3.100  -6.937  0.12191        0.258
## income      1999    0.701 -0.01274  0.179   0.715 -0.19903        0.357
## education   1999    0.485  0.00699  0.139   0.481  0.00321        0.678
```

The summary gives the original sample value for each component of the bootstrapped statistics, along with the bootstrap estimates of bias, the difference $\overline{T}^* - T$ between the average bootstrapped value of the statistic and its original-sample value. The bootstrap estimates of standard error $[\widehat{\mathrm{SE}}^*(T^*)]$ are computed as the standard deviation of the bootstrap replicates. As explained in the previous section, these values may be used to construct normal-theory confidence intervals for the regression coefficients. In this example the bootstrap standard errors of the `income` and `education` coefficients are substantially larger than the asymptotic estimates reported by `rlm`.

[5]The time will vary slightly from run to run, and more substantially depending on hardware and operating system.

The bootstrap estimates of skewness and kurtosis are added here because `high.moments=TRUE`; the default is `FALSE`. See `help(summary.boot)` for additional arguments to the `summary` method for `"Boot"` objects.

The function `vcov` will return the estimated covariance matrix, that is the sample covariance matrix of the bootstrap samples,

```
vcov(duncan.boot)

##              (Intercept)   income education
## (Intercept)      9.6105 -0.02660  -0.10225
## income          -0.0266  0.03188  -0.02322
## education       -0.1022 -0.02322   0.01931
```

This quantity may be needed in some futher computations,

The **car** package also includes a `confint` method to produce confidence intervals for `"boot"` objects:

```
confint(duncan.boot, level=.90, type="norm")

## Bootstrap quantiles, type =  normal
##
##                   5 %   95 %
## (Intercept) -12.3495 -2.151
## income        0.4205  1.008
## education     0.2499  0.707

confint(duncan.boot, parm=2:3, level=c(.68, .90, .95), type="perc")

## Bootstrap quantiles, type =  percent
##
##           2.5 %    5 %   16 %   84 %   95 % 97.5 %
## income    0.316 0.3764 0.4961 0.8441 0.9539 1.0192
## education 0.221 0.2799 0.3702 0.6317 0.7297 0.7795

confint(duncan.boot, level=.95, type="bca")

## Bootstrap quantiles, type =  bca
##
##                 2.5 %  97.5 %
## (Intercept) -12.9587 -1.3133
## income        0.2210  0.9415
## education     0.2743  0.8311
```

The first of these examples uses normal theory with the bootstrap standard errors. The second example uses the percentile method, and gives the quantiles for a number of intervals simultaneously. The final example uses the BCa method, and this is the default if no arguments beyond the first are provided in the call to the function.

A graphical view of the bootstraps can be obtained with the `hist` function, as shown in Figure **??**, drawn by

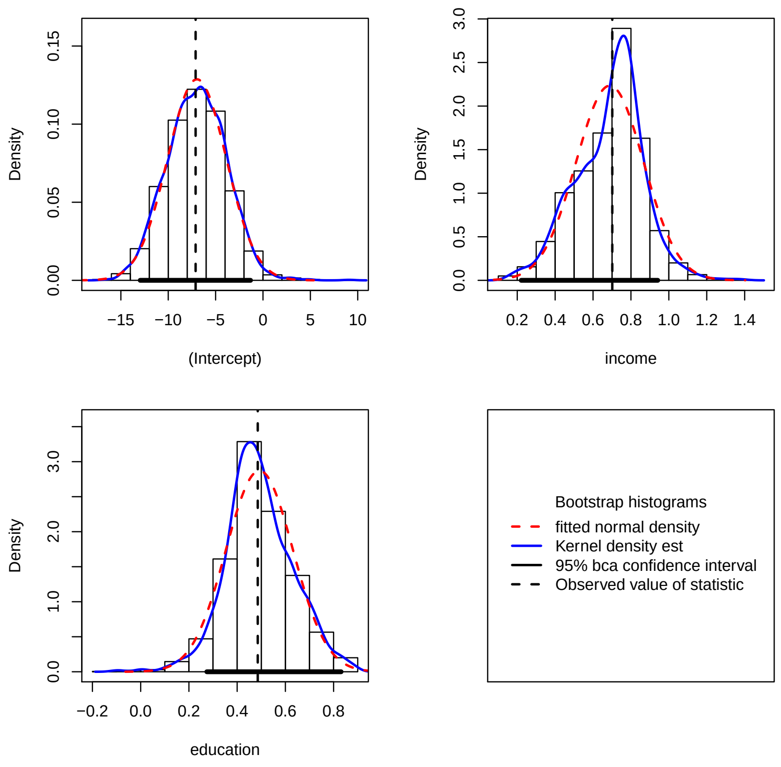


Figure 1: Case bootstrap histograms.

```
hist(duncan.boot, legend="separate")
```

There is a separate histogram for each bootstrapped quantity, here each coefficient. In addition to the histograms we also get kernel density estimates and the normal density based on the bootstrap mean and standard deviation. The vertical dashed line makes the original point-estimate, and the thick horizontal line gives a confidence interval based on the bootstrap. Whereas the two density estimates for the intercept are similar, the normal approximation is poor for the other coefficients, and confidence intervals are not close to symmetric about the original values. This suggests that inference from the bootstrap is different from the asymptotic theory, and that the bootstrap is likely to be more accurate in this small sample. See `help("hist.boot")` for additional arguments to `hist`.

We next use the `dataEllipse` function from the **car** package to examine the joint distribution of the bootstrapped `income` and `education` coefficients. The function draws a scatterplot of the pairs of coefficients, with concentration ellipses superimposed (Figure 2):

```
dataEllipse(duncan.boot$t[, 2], duncan.boot$t[, 3],
    xlab="income coefficient", ylab="education coefficient",
    cex=0.3, levels=c(.5, .95, .99), robust=TRUE)
```

The first two arguments to `dataEllipse` are `duncan.boot$t[, 2]` and `duncan.boot$t[, 3]`, which are the vectors of bootstraps for the second and third coefficients, for `income` and `education`.

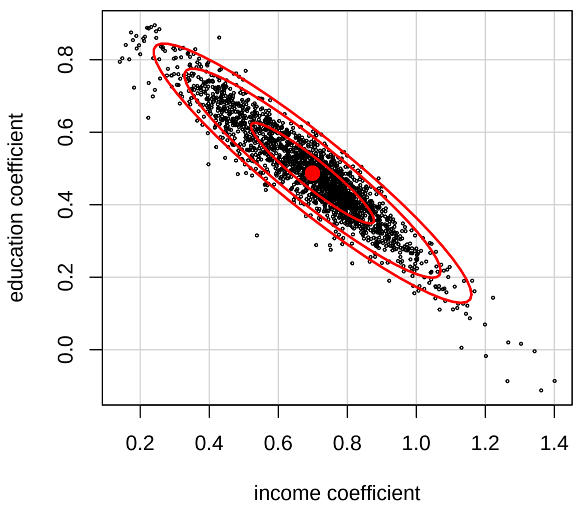


Figure 2: Scatterplot of bootstrap replications of the `income` and `education` coefficients from the Huber regression for Duncan's occupational-prestige data. The concentration ellipses are drawn at the 50, 90, and 99% levels using a robust estimate of the covariance matrix of the coefficients.

3.2 Additional Functionality from the boot Package

The objects created when using the `Boot` function can also be examined with all the helper functions that are included in the **boot** package. For example, if you simply print the object:

```
library(boot)

##
## Attaching package:  'boot'
## The following object is masked from 'package:car':
##
##     logit

duncan.boot

##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot::boot(data = dd, statistic = boot.f, R = R, .fn = f)
##
##
## Bootstrap Statistics :
##      original    bias    std. error
## t1*  -7.1107  0.139653      3.1001
## t2*   0.7014 -0.012739      0.1785
## t3*   0.4854  0.006993      0.1390
```

the resulting output is from the `print` method provided by the **boot** package for `"boot"` objects. This is similar to the `summary.boot` method from **car**, but the labels are less informative. The `boot.array` function returns an $R \times n$ matrix in which the entry in row b, column i indicates how many times the ith observation appears in the bth bootstrap sample:

```
duncan.array <- boot.array(duncan.boot)
duncan.array[1:2, ]

##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]
## [1,]    0    2    1    2    0    0    1    0    2     1     3     2     3     1
## [2,]    2    2    1    1    2    0    0    0    0     2     2     1     0     2
##      [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25] [,26]
## [1,]     2     0     1     1     1     2     1     1     0     1     0     1
## [2,]     1     3     1     0     0     0     0     0     0     2     2     1
##      [,27] [,28] [,29] [,30] [,31] [,32] [,33] [,34] [,35] [,36] [,37] [,38]
## [1,]     3     2     0     0     1     1     1     0     0     2     1     0
## [2,]     0     1     0     2     0     2     1     2     3     0     2     1
##      [,39] [,40] [,41] [,42] [,43] [,44] [,45]
## [1,]     0     0     2     2     1     0     0
## [2,]     1     1     1     1     1     0     1
```

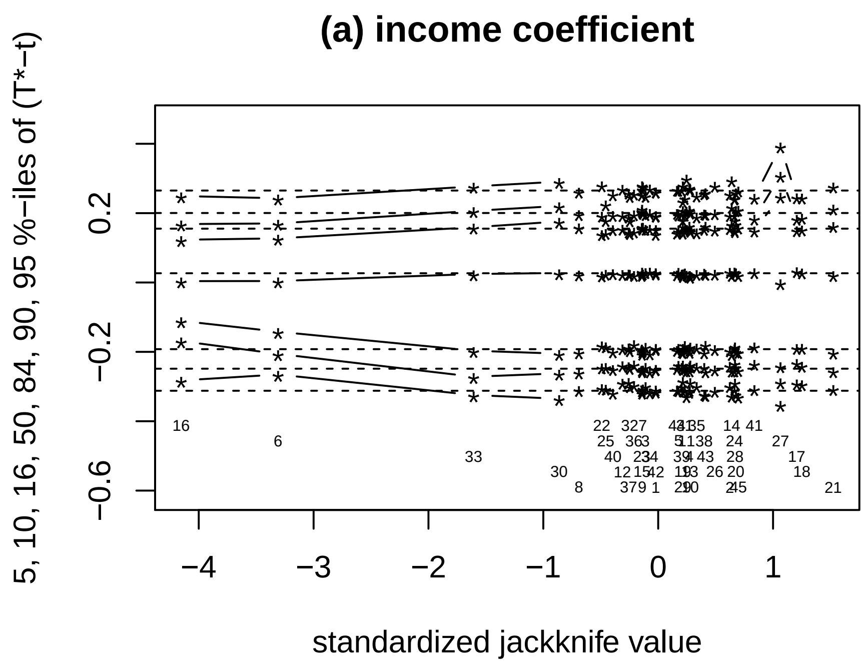
Thus, for example, observation 1 appears twice in the second bootstrap sample, but not at all in the first sample.

The (unfortunately named) `jack.after.boot` function displays a diagnostic *jackknife-after-bootstrap* plot. This plot shows the sensitivity of the statistic and of the percentiles of its bootstrapped distribution to deletion of individual observations. An illustration, for the coefficients of `income` and `education`, appears in Figure 3, which is produced by the following commands:

```
par(mfcol=c(2, 1))
jack.after.boot(duncan.boot, index=2, main="(a) income coefficient")
jack.after.boot(duncan.boot, index=3, main="(b) education coefficient")
```

The horizontal axis of the graph, labeled "standardized jackknife value," is a measure of the influence of each observation on the coefficient. The observation indices corresponding to the points in the graph are shown near the bottom of the plot. Thus observations 6 and 16 serve to decrease the `income` coefficient and increase the `education` coefficient. As is familiar from Chapters 1 and 6 of the *R Companion*, these are the problematic occupations minister and railroad-conductor.

The horizontal broken lines on the plot are quantiles of the bootstrap distribution of each coefficient, centered at the value of the coefficient for the original sample. By default the .05, .10, .16, .50, .84, .90, and .95 quantiles are plotted. The points connected by solid lines show the quantiles estimated only from bootstrap samples in which each observation in turn did not appear. Therefore, deleting the occupation minister or conductor makes the bootstrap distributions for the coefficients slightly less dispersed. On the other hand, removing occupation 27 (railroad-engineer) or 30 (plumber) makes the bootstrap distributions somewhat more variable. From our earlier work on Duncan's data (see, in particular, Chapter 6), we recognize railroad-engineer as a high-leverage



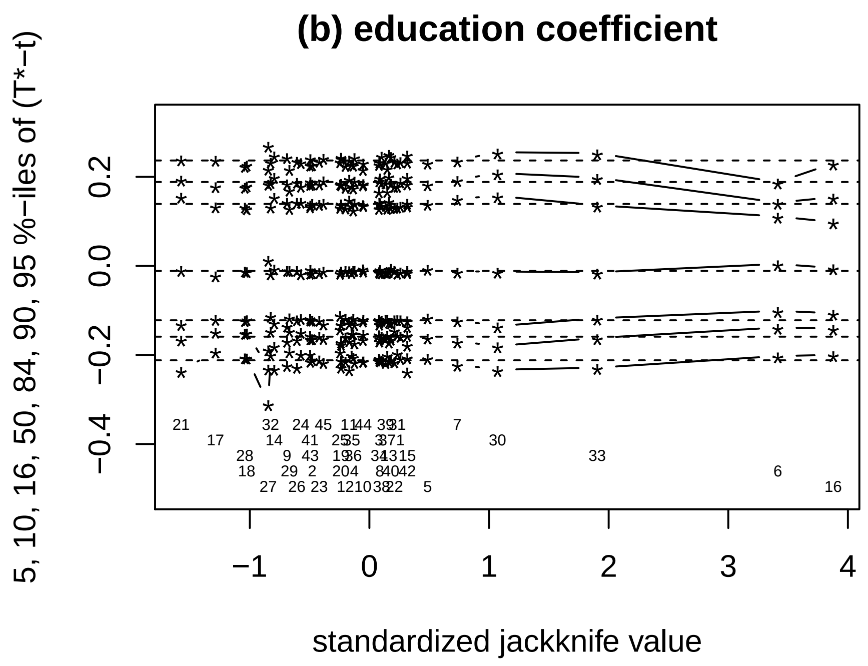


Figure 3: Jackknife-after-bootstrap plot for the `income` (a) and `education` (b) coefficients in the Huber regression for Duncan's occupational-prestige data.

but in-line occupation; it is unclear to us, however, why the occupation plumber should stand out in this manner.

boot objects have their own `plot` method that differs from the `hist` method described above, a method called `boot.ci` that is useful for comparing methods for generating confidence intervals but less useful for data analysis, and several other methods; see `help("boot")`.

3.3 Bypassing the `Boot` Function

In this section we show how to use `boot` directly to get a bootstrap. For regression problems in which the data are sampled independently, this will generally be unnecessary as `Boot` provides sufficient functionality, but for other problems, for example, a bootstrap for the distribution of a sample median. You will also have to call `boot` directly when the data are generated from a complex sampling design, a subject to which we return briefly in Section 6. For these purposes you will need to write a function that lays out what needs to be done for each bootstrap sample. Here is an example that corresponds to what `Boot` does by default:

```
boot.huber <- function(data, indices, maxit=20){
    data <- data[indices, ]  # select obs. in bootstrap sample
    mod <- rlm(prestige ~ income + education, data=data, maxit=maxit)
    coef(mod)  # return coefficient vector
}
```

This function has three arguments. The first argument takes a data frame. The second argument is a vector of row indices that make up the bootstrap sample, and will be supplied for each bootstrap replication by `boot`. The function `boot.huber` takes a third argument, `maxiter`, which sets the maximum number of iterations to perform in each M-estimation; we included this provision because we found that the default of 20 iterations in `rlm` is not always sufficient. The `boot` function is able to pass additional arguments through to the function specified in its `statistic` argument. This function will recompute the regression fit for each bootstrap sample, and then return the coefficient estimates.

Here is the call to `boot`

```
set.seed(12345) # for reproducibility
library(boot)  # needed only if boot not previously loaded
duncan.boot.1 <- boot(data=Duncan, statistic=boot.huber,
   R=1999, maxit=200)
```

The first argument to `boot` is the data set—a vector, matrix, or data frame—to which bootstrap resampling is to be applied. Each element of the data vector, or each row of the matrix or data frame, is treated as an observation. The argument `statistic` is function that returns the possibly vector-valued statistic to be bootstrapped. The argument `R` is the number of bootstrap replications. The `maxit` argument is passed to the `statistic` function. Additional arguments are described in the on-line help for `boot`.

Since the same random seed was used in creating `duncan.boot.1` using `boot` and `duncan.boot` using `Boot`, the returned objects are identical, and you can use the `hist`, `summary` and `confint` methods with either object.

3.4 Fixed-x or Residual Resampling

The observations in Duncan's occupational-prestige study are meant to represent a larger population of all Census occupations, but they are not literally sampled from that population. It therefore makes some sense to think of these occupations, and hence the pairs of `income` and `education` values used in the regression, as fixed with respect to replication of the study. The response values, however, are random, because of the error component of the model. There are other circumstances in which it is even more compelling to treat the predictors in a study as fixed — for example, in a designed experiment where the values of the predictors are set by the experimenter.

How can we generate bootstrap replications when the model matrix $\mathbf{X}$ is fixed? In *residual resampling*, we write

$$\begin{aligned}\mathbf{Y} &= \mathbf{X}\widehat{\boldsymbol{\beta}} + (\mathbf{Y} - \mathbf{X}\widehat{\boldsymbol{\beta}}) \\ &= \widehat{\mathbf{Y}} + \mathbf{e}\end{aligned}$$

so $\widehat{\mathbf{Y}}$ is the vector of fitted values and $\mathbf{e}$ is a vector of residuals. In residual resampling we fix $\widehat{\mathbf{Y}}$ and resample the residuals $\mathbf{e}$, to get

$$\mathbf{Y}^* = \widehat{\mathbf{Y}} + \mathbf{e}^*$$

The `"residual"` method for `Boot` implements a slight modification of this procedure, by resampling scaled and centered residuals, with i-th element

$$r_i = \frac{e_i}{\sqrt{1 - h_i}} - \bar{r}$$

where h_i is the i-leverage. This is the suggested method in Davison and Hinkley (1997, Alg. 6.3, p. 271).

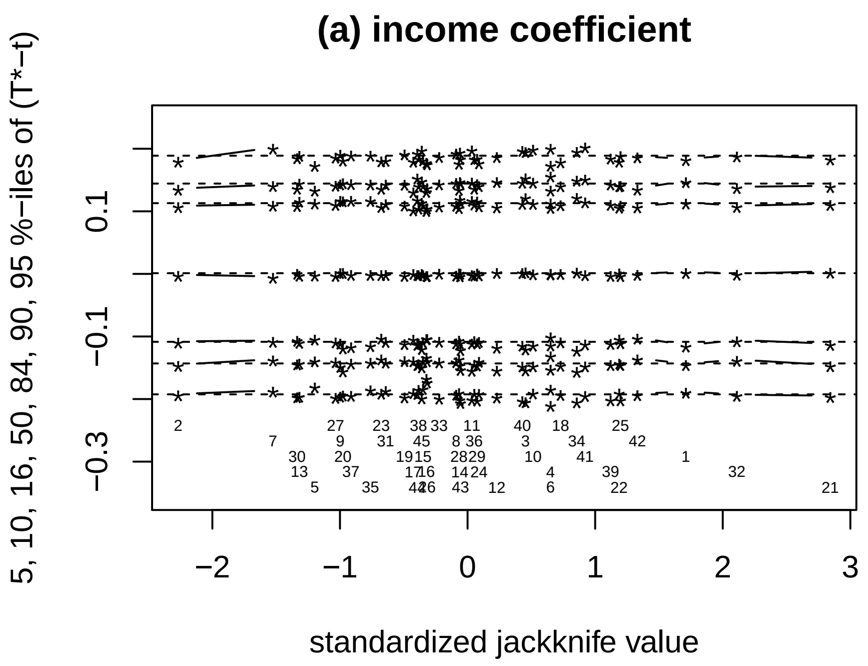
```
set.seed(54321) # for reproducibility
summary(duncan.fix.boot <- Boot(mod.duncan.hub, R=1999, method="residual"))

##                R original  bootBias bootSE bootMed
## (Intercept) 1999   -7.111 -0.047192 3.9041  -7.090
## income      1999    0.701 -0.000903 0.1154   0.702
## education   1999    0.485  0.001835 0.0937   0.487
```

Examining the jackknife-after-bootstrap plot for the fixed-x resampling results (Figure 4) provides some insight into the properties of the method:

```
par(mfcol=c(2, 1))
jack.after.boot(duncan.fix.boot, index=2, main="(a) income coefficient")
jack.after.boot(duncan.fix.boot, index=3, main="(b) education coefficient")
```

The quantile traces in Figure 4 are much less variable than in Figure 3 for random-x resampling, because in fixed-x resampling residuals are decoupled from the original observations. In effect, fixed-x resampling enforces the assumption that the errors are identically distributed by resampling residuals from a common distribution. Consequently, if the model is incorrectly specified — for example, if there is unmodeled nonlinearity, non-constant error variance, or outliers — these characteristics will not carry over into the resampled data sets. For this reason, it may be preferable to perform random-x resampling even when it makes sense to think of the model matrix as fixed.

`Boot` does not allow `method="residual"` for generalized linear models; see Davison and Hinkley (1997) for a discussion of the methodology, and its problems.



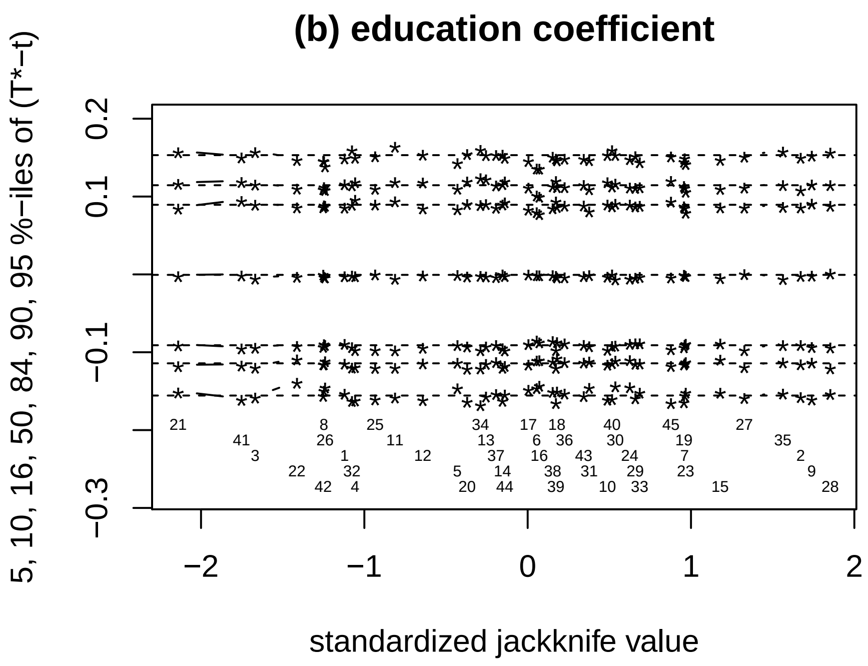


Figure 4: Jackknife-after-bootstrap plot for the `income` (a) and `education` (b) coefficients in the Huber regression for Duncan's occupational-prestige data, using fixed-x resampling.

4 Bootstrap Hypothesis Tests

Tests for individual coefficients equal to zero can be found by inverting a confidence interval: if the hypothesized value does not fall in a 95% confidence interval, for example, then the significance level of the test is less than $(100 - 95)/100 = 0.05$.

We will consider one specific testing problem. Imagine that in Duncan's regression, we want to use the robust-regression estimator to test the hypothesis that the `income` and `education` coefficients are the same, H_0: $\beta_1 = \beta_2$. This hypothesis arguably makes some sense, because both predictors are scaled as percentages. We could test the hypothesis with the Wald statistic

$$z = \frac{b_1 - b_2}{\left[(0, 1, -1)\widehat{\text{Var}}(\mathbf{b}) \begin{pmatrix} 0 \\ 1 \\ -1 \end{pmatrix} \right]^{1/2}}$$

where $\mathbf{b}$ is the vector of estimated regression coefficients; b_1 and b_2 are respectively the `income` and `education` coefficients; and $\widehat{\text{Var}}(\mathbf{b})$ is the estimated asymptotic covariance matrix of the coefficients. If we can trust the asymptotic normality of $\mathbf{b}$ and its asymptotic covariance matrix, then z is distributed as a standard normal variable under the null hypothesis. The numerator and denominator of z are easily computed with the **car** function `deltaMethod`:[6]

```
(d <- deltaMethod(mod.duncan.hub, "income - education"))

##                      Estimate    SE   2.5 % 97.5 %
## income - education    0.216 0.184 -0.1446 0.5766
```

The output from `deltaMethod` is a data frame with one row and two columns, so z is then

```
(z.diff <-  d[1, 1] / d[1, 2])

## [1] 1.174
```

The function `deltaMethod` can be used for any linear or nonlinear combination of the coefficients. The corresponding significance level of the test, assuming asymptotic normality for z, is

```
c(two.tail.p=2*pnorm(z.diff, lower.tail=FALSE))

## two.tail.p
##     0.2404
```

In a small sample such as this, however, we may be more comfortable relying on the bootstrap to get a significance level.

This test is easily computed: To use `Boot`, we write a function `f.diff`:

[6]The `linearHypothesis` function in the **car** package fails for this model because objects produced by `rlm` inherit from class `"lm"` and the `"lm"` method for `linearHypothesis` does not work with `"rlm"` objects; one could, however, apply the default method `car:::linearHypothesis.default` to produce the same results as `deltaMethod`.

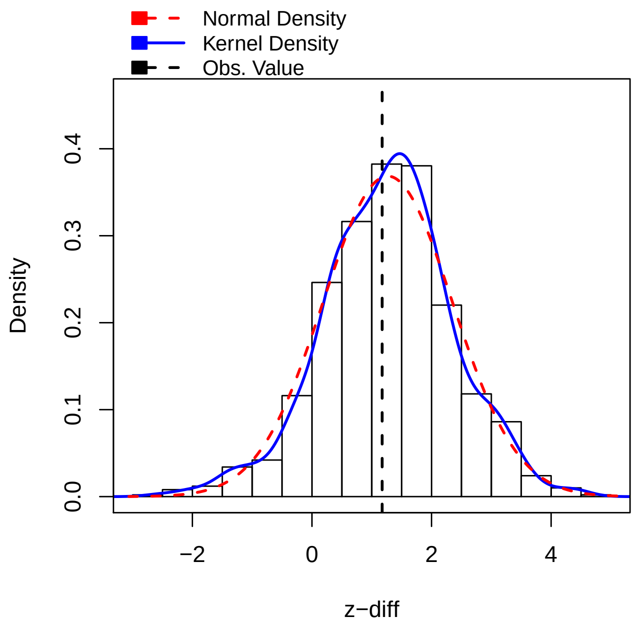


Figure 5: Distribution of the bootstrapped test statistic z^* for the hypothesis that the coefficients of `income` and `education` are equal.

```
f.diff <- function(mod){
   d <- deltaMethod(mod, "income-education")
   d[1, 1]/d[1, 2]
   }
```

and then the call to `Boot` is

```
set.seed(2468) # for reproducibility
boot.diff <- Boot(mod.duncan.hub, R=999, f=f.diff,
  labels="z-diff", method="residual")
hist(boot.diff, ci="none")
```

The histogram is shown in Figure 5. More of the bootstrap density is to the right of the observed value of z than is modeled by fitting a normal distribution,

The two-tailed p-value based on this bootstrap is estimated by the fraction of bootstrap values $|z^*| > |z|$, which we compute as

```
R <- 1999
c(bootp =
  (1 + sum(abs(boot.diff$t[, 1]) > abs(boot.diff$t0[1])))
  / ( R + 1))

##  bootp
## 0.2855
```

In this expression `boot.diff$t[, 1]` is the vector of bootstrapped values z^* and `boot.diff$t0[1]` is the observed value z. We added one to the numerator and denominator to improve accuracy.

Testing in general using the bootstrap is potentially complex and beyond the purpose of this appendix. We recommend Davison and Hinkley (1997, Sec. 6.3.2) for a discussion of testing in the regression context.

5 Using `Boot` With Other Regression Modeling Functions

The `Boot` function was originally written specifically for regression models similar to a `glm` model, like the `rlm` function used earlier as an example. Thanks to Achim Zeileis, the class of regression models that can be used is greatly expanded. We will illustrate by fitting using the **betareg** package for fitting regression with beta-distributed response,[7]

```
library("betareg")
data("ReadingSkills", package = "betareg")
m <- betareg(accuracy ~ iq * dyslexia | iq + dyslexia, data = ReadingSkills)
summary(m)

##
## Call:
## betareg(formula = accuracy ~ iq * dyslexia | iq + dyslexia, data = ReadingSkills)
##
## Standardized weighted residuals 2:
##    Min     1Q Median     3Q    Max
## -2.390 -0.642  0.157  0.852  1.645
##
## Coefficients (mean model with logit link):
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)    1.123      0.143    7.86  3.7e-15
## iq             0.486      0.133    3.65  0.00026
## dyslexia      -0.742      0.143   -5.20  2.0e-07
## iq:dyslexia   -0.581      0.133   -4.38  1.2e-05
##
## Phi coefficients (precision model with log link):
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)    3.304      0.223   14.84  < 2e-16
## iq             1.229      0.267    4.60  4.2e-06
## dyslexia       1.747      0.262    6.66  2.8e-11
##
## Type of estimator: ML (maximum likelihood)
## Log-likelihood: 65.9 on 7 Df
## Pseudo R-squared: 0.576
## Number of iterations: 25 (BFGS) + 1 (Fisher scoring)
```

The `betareg` function fits two linear predictors, one for a mean model, and a second linear predictor for a precision model. The response `accuracy` is a rate between zero and one. The

[7]You may need to install the `betareg` package to reproduce this example.

predictors are an indicator for `dysleia` and a continuous predictor `iq` of non-verbal IQ. The beta distribution has two parameters that can be viewed as a mean parameter and a precision parameter, and hence the need for two linear predictors. The bootstrap can be used estimate the precision of the estimates.

To apply the `Boot` function, the following functions must exist and return values for the regression object

1. The object must have an `update` method, as do most modeling functions

2. The function that created the object must have a `subset` argument, as do most functions that have a `data` argument.

3. For an object named `m`, the call `residuals(m, type="pearson")` must return a vector of residuals. If this is not the case, and you have another function (say, `myres`) that returns Pearson residuals, you can create a method. First, find the class of the object,

   ```
   class(m)

   ## [1] "betareg"
   ```

   Then create the method:

   ```
   residuals.betareg <- function(object, type="pearson") {myres(object)}
   ```

   This is unnecessary for `betareg` objects, as it already has a `residuals` method.

4. For an object named `m`, the call `fitted(m)` must return a vector of fitted values. This function exists for `betareg` objects; if it didn't you would need to write a `fitted.betareg` method, if `m` were of class `betareg`. This is unnecessary for `betareg` objects since the required `fitted` method already exists.

5. The function call `hatvalues(m)` must return a vector of leverages, also called hat values. This is used to improve the performance of the bootstrap, but simply returning the value 1 will work. If your regression function does not have a `hatvalues` method, we suggest using, for a regression object of class `myobj`,

   ```
   hatvalues.myobj <- function(object) 1
   ```

Since `betareg` has all the requisite functions, the use of `Boot` is straightforward:

```
b <- Boot(m, R = 250)
sqrt(diag(vcov(b)))

##       (Intercept)                iq          dyslexia       iq:dyslexia
##            0.1918            0.1780            0.1928            0.1900
## (phi)_(Intercept)          (phi)_iq    (phi)_dyslexia
##            0.3103            0.6375            0.3877
```

This has returned the bootstrapped standard errors.

In this second example we illustrate passing additional arguments ot `boot` using an example from the **crch** package that fits censored regression with conditional heteroscedasticy. See the help for the package and for the data set for more details about the models fit.

```
library("crch")
## data pre-processing
data("RainIbk", package = "crch")
RainIbk$sqrtensmean <- apply(sqrt(RainIbk[,grep('^rainfc',names(RainIbk))]), 1, mean)
RainIbk$sqrtenssd <- apply(sqrt(RainIbk[,grep('^rainfc',names(RainIbk))]), 1, sd)
m <- crch(sqrt(rain) ~ sqrtensmean | sqrtenssd, data = RainIbk,
  dist = "logistic", left = 0)
```

We fit a bootstrap three times, each using the same starting value for random numbers, to illustrate reducing processing time.

```
set.seed(1); system.time(b1 <- Boot(m, R = 250))

##    user  system elapsed
##   45.39    0.02   45.42

set.seed(1); system.time(b2 <- Boot(m, R = 250, start = TRUE))

##    user  system elapsed
##   28.23    0.03   28.27

set.seed(1); system.time(b3 <- Boot(m, R = 250, start = TRUE,
  parallel = "multicore", ncpus = 2))

##    user  system elapsed
##   29.26    0.00   29.31

sapply(list(b1, b2, b3), function(b) sqrt(diag(vcov(b))))

##                         [,1]    [,2]    [,3]
## (Intercept)          0.07071 0.07070 0.07070
## sqrtensmean          0.01971 0.01971 0.01971
## (scale)_(Intercept) 0.04146 0.04146 0.04146
## (scale)_sqrtenssd   0.03116 0.03116 0.03116
```

6 Concluding Remarks

Extending random-x resampling to other sorts of parametric regression models, such as generalized linear models, is straightforward. In many instances, however, fixed-x resampling requires special treatment, as does resampling for nonparametric regression.

The discussion in the preceding sections assumes independent random sampling, but bootstrap methods can easily be adapted to other sampling schemes. For example, in stratified sampling, bootstrap resampling is simply performed within strata, building up a bootstrap sample much as the original sample was composed from subsamples for the strata. Likewise, in a cluster sample, we resample clusters rather than individual observations. If the elements of the sample were selected with unequal probability, then so must the elements of each bootstrap sample.

The essential point is to preserve the analogy between the selection of the original sample from the population and the selection of each bootstrap sample from the original sample. Indeed, one of the attractions of the bootstrap is that it can provide correct statistical inference for complex sampling designs which often are handled by ad-hoc methods.[8] The software in the **boot** package can accommodate these complications; see, in particular, the `stype` and `strata` arguments to the `boot` function.

7 Complementary Reading and References

Efron and Tibshirani (1993) and Davison and Hinkley (1997) provide readable book-length treatments of the bootstrap. For shorter presentations, see Fox (2008, chap. 21), Weisberg (2005, sec. 4.6) and Stine (1990).

References

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

Efron, B. (1979). Bootstrap methods: another look at the jackknife. *Annals of Statistics*, 7:1–26.

Efron, B. and Tibshirani, R. J. (1993). *An Introduction to the Bootstrap.* Chapman and Hall, New York.

Fox, J. (2008). *Applied Regression Analysis and Generalized Linear Models.* Sage, Thousand Oaks, CA, second edition.

Fox, J. and Weisberg, S. (2011). *An R Companion to Applied Regression.* Sage, Thousand Oaks, CA, second edition.

Lumley, T. (2010). *Complex Surveys: A Guide to Analysis Using R.* John Wiley & Sons, Hoboken, NJ.

Stine, R. (1990). An introduction to bootstrap methods: examples and ideas. In Fox, J. and Long, J. S., editors, *Modern Methods of Data Analysis*, pages 325–373. Sage, Newbury Park, CA.

Weisberg, S. (2005). *Applied Linear Regression.* John Wiley & Sons, Hoboken, NJ, third edition.

[8]The **survey** package for R (Lumley, 2010) has extensive facilities for statistical inference in complex sample surveys.