Sintaxis: boot y boot.ci

Pedro Luque

Índice

1. Bootstrap Resampling (remuestreo Bootstrap): boot

1.1. Descripción

Genera réplicas de bootstrap de R de un estadístico aplicada a unos datos. Son posibles tanto el remuestreo paramétrico como no paramétrico.

- Para el **bootstrap no paramétrico**, los posibles métodos de remuestreo son el bootstrap ordinario, el bootstrap balanceado, el remuestreo antitético y la permutación.
- Para **problemas no paramétricos de múltiples muestras se usa remuestreo estratificado**: esto se especifica incluyendo un vector de estratos en la llamada a boot. Pueden especificarse los pesos de importancia del remuestreo.

1.2. Sintaxis

```
boot(data, statistic, R, sim = "ordinary", stype = c("i", "f", "w"),
     strata = rep(1,n), L = NULL, m = 0, weights = NULL,
     ran.gen = function(d, p) d, mle = NULL, simple = FALSE, ...,
     parallel = c("no", "multicore", "snow"),
     ncpus = getOption("boot.ncpus", 1L), cl = NULL)
```

1.3. Argumentos

- `data`: Los datos como vector, matriz o marco de datos. Si es una matriz o un marco de datos, cada fila se considera una observación multivariante.
- `statistic`: Una función que, cuando se aplica a los datos, devuelve un vector que contiene las estadísticas de interés. Cuando `sim = "parametric"`, el primer argumento de `statistic` deben ser los datos. Para cada réplica, se pasará un conjunto de datos simulado devuelto por `ran.gen`.

  En todos los demás casos, `statistic` debe tener al menos dos argumentos.

  - El primer argumento que se pase siempre serán los datos originales.
  - El segundo será un vector de índices, frecuencias o ponderaciones que definen la muestra bootstrap.
  - Además, si se requieren predicciones, entonces se requiere un tercer argumento que sería un vector de los índices aleatorios utilizados para generar las predicciones bootstrap.
  - Cualquier otro argumento se puede pasar a `statistic` a través del argumento `...`
- `R`: El número de replicaciones de bootstrap. Por lo general, este será un único entero positivo. Para el remuestreo de importancia, algunos remitentes pueden usar un conjunto de pesos y otros usan un conjunto diferente de pesos. En este caso, sería un vector de enteros donde cada componente le da al número de resamitaciones de cada una de las filas de pesos.
- `sim`: Una cadena de caracteres que indica el tipo de simulación requerida. Los valores posibles son "ordinary" (el valor predeterminado), "parametric", "balanced", "permutation", o "antithetic". El remuestreo de importancia se especifica al incluir los pesos de importancia; El tipo de remuestreo de importancia aún debe especificarse, pero puede ser solo "ordinary" o "balanced" en este caso.
- `stype`: Una cadena de caracteres que indica lo que representa el segundo argumento de la estadística. Los posibles valores de stype son "i" (índices, el valor predeterminado), "f" (Frecuencias), o "w" (pesos). No se utiliza para `sim = "parametric"`.
- `strata`: Un vector entero o factor que especifica los estratos para problemas de múltiples muestras. Esto puede especificarse para cualquier simulación, pero se ignora cuando Sim = "paramétrico". Cuando se suministra estratos para un bootstrap no paramétrico, las simulaciones se realizan dentro de los estratos especificados.
- `L`: Vector de valores de influencia evaluados en las observaciones. Esto se usa solo cuando `sim` es "antithetic". Si no se suministra, se calculan a través de una llamada a `empinf`. Esto utilizará el "jacknife infinitesimal", siempre que `stype` sea "w", de lo contrario se usa el "jacknife" habitual.
- `m`: El número de predicciones que se deben hacer en cada replicación de bootstrap. Esto es más útil para los modelos lineales (generalizados). Esto solo se puede utilizar cuando `sim` es "ordinary". `m`, por lo general, será un entero escalar, pero si hay estratos, puede ser un vector con longitud igual al número de estratos, lo que especifica cuántos de los errores para la predicción deben provenir de cada estrato. Las predicciones reales deben devolverse como la parte final de la salida de `statistic`, que también debe tomar un argumento que le da al vector de los índices de los errores que se utilizarán para las predicciones.
- `weights`: Vector o matriz de pesos de importancia. Si es un vector, entonces debería tener tantos elementos como observaciones hay en los datos. Cuando se requiere simulación de más de un conjunto de pesos, los pesos deben ser una matriz donde cada fila de la matriz es un conjunto de pesos de importancia. Si los pesos son una matriz, entonces `R` debe ser un vector de longitud `nrow(weights)`. Este parámetro se ignora si `sim` no es "ordinary" o "balanced".
- `ran.gen`: Esta función se usa solo cuando `sim = "parametric"` cuando describe cómo se generarán los valores aleatorios. Debería ser una función de dos argumentos. El primer argumento deben ser los datos observados y el segundo argumento consiste en cualquier otra información necesaria (por ejemplo, estimaciones de parámetros). El segundo argumento puede ser una lista, lo que permite pasar cualquier número de elementos a `ran.gen`. El valor devuelto debe ser un conjunto de datos simulados de la misma forma que los datos observados que se pasarán a `statistic` para obtener una réplica de arranque. Es

importante que el valor devuelto tenga la misma forma y tipo que el conjunto de datos original. Si no se especifica `ran.gen`, el valor predeterminado es una función que devuelve los datos originales, en cuyo caso toda la simulación debe incluirse como parte de la estadística. El uso de `sim = "parametric"` con un `ran.gen` adecuado permite al usuario implementar cualquier tipo de remuestreo no paramétrico que no sea compatible directamente.

- `mle`: El segundo argumento que se pasará a `ran.gen`. Normalmente, estas serán estimaciones de máxima probabilidad de los parámetros. Por razones de eficiencia, mle es a menudo una lista que contiene todos los objetos que necesita ran.gen y que puede calcularse utilizando únicamente el conjunto de datos original.
- `simple`: lógico, solo se permite que sea TRUE para `sim = "ordinary"`, `stype = "i"`, `n = 0` (de lo contrario, se ignora con una advertencia). De forma predeterminada, se crea una matriz de índice `n por R`: esta puede ser grande y si `simple = TRUE` esto se evita muestreando por separado para cada replicación, que es más lenta pero usa menos memoria.
- `...`: Otros argumentos con nombre para `statistic` que se pasan sin cambios cada vez que se llama. Cualquiera de estos argumentos a la estadística debe seguir los argumentos que `statistic` debe tener para la simulación. Tenga cuidado con la coincidencia parcial con los argumentos de arranque enumerados anteriormente, y que los argumentos denominados `X` y `FUN` causan conflictos en algunas versiones de arranque (pero no en esta).
- `parallel`: El tipo de operación en paralelo que se utilizará (si corresponde). Si falta, el valor predeterminado se toma de la opción "boot.parallel" (y si no está configurado, "no").
- `ncpus`: integer: Número de procesos que se utilizarán en operación paralela: normalmente, uno elegiría esto al número de CPU disponibles.
- `cl`: Un clúster `parallel` o `snow` para usar si `parallel = "snow"`. Si no se proporciona, se crea un clúster en la máquina local durante la duración de la llamada a `boot`.

1.4. Detalles

El `statistic` será "bootstrapped" puede ser tan simple o complicada como se desee, siempre y cuando sus argumentos correspondan al conjunto de datos y (para un bootstrap no paramétrico) un vector de índices, frecuencias o pesos. El `statistic` se trata como una caja negra por la función de arranque y no se verifica para asegurarse de que se cumplan estas condiciones.

La primera vez que se describe el "balanced bootstrap" es en Davison, Hinkley y Schechtman (1986). El "antithetic bootstrap" es descrito por Hall (1989) y es experimental, especialmente cuando se usa con estratos. Los otros tipos de simulación no paramétricos son el bootstrap ordinario (posiblemente con probabilidades desiguales) y la permutación que devuelve permutaciones aleatorias de casos. Todos estos métodos trabajan independientemente dentro de estratos si se suministra ese argumento.

Para el bootstrap paramétrico, es necesario que el usuario especifique cómo se debe realizar el remuestreo. La mejor manera de lograr esto es especificar la función `ran.gen`, que devolverá un conjunto de datos simulado desde el conjunto de datos observado y un conjunto de estimaciones de parámetros especificadas en `mle`.

1.5. Valores devueltos

El valor devuelto es un objeto de la clase "boot", que contiene los siguientes componentes:

- `t0`: The observed value of statistic applied to data.
- `t`: A matrix with sum(R) rows each of which is a bootstrap replicate of the result of calling statistic.
- `R`: The value of R as passed to boot.
- `data`: The data as passed to boot.
- `seed`: The value of .Random.seed when boot started work.

- `statistic`: The function statistic as passed to boot.
- `sim`: Simulation type used.
- `stype`: Statistic type as passed to boot.
- `call`: The original call to boot.
- `strata`: The strata used. This is the vector passed to boot, if it was supplied or a vector of ones if there were no strata. It is not returned if sim is "parametric".
- `weights`: The importance sampling weights as passed to boot or the empirical distribution function weights if no importance sampling weights were specified. It is omitted if sim is not one of "ordinary" or "balanced".
- `pred.i`: If predictions are required (m > 0) this is the matrix of indices at which predictions were calculated as they were passed to statistic. Omitted if m is 0 or sim is not "ordinary".
- `L`: The influence values used when sim is "antithetic". If no such values were specified and stype is not "w" then L is returned as consecutive integers corresponding to the assumption that data is ordered by influence values. This component is omitted when sim is not "antithetic".
- `ran.gen`: The random generator function used if sim is "parametric". This component is omitted for any other value of sim.
- `mle`: The parameter estimates passed to boot when sim is "parametric". It is omitted for all other values of sim.

There are c, plot and print methods for this class.

1.6. Operaciones en paralelo

Cuando se usa paralelo = "multinúcleo" (no disponible en Windows), cada proceso de trabajo hereda el entorno de la sesión actual, incluido el espacio de trabajo y los espacios de nombres cargados y los paquetes adjuntos (pero no la semilla de número aleatorio: ver más abajo).

Se necesita más trabajo cuando se usa paralelo = "nieve": los procesos de trabajo son procesos R recién creados, y las estadísticas deben organizarse para configurar el entorno que necesita: a menudo, una buena manera de hacerlo es hacer uso del alcance léxico ya que cuando la estadística se envía a los procesos de trabajador, también se envía su entorno adjunto. (Por ejemplo, vea el ejemplo de jack.after.boot donde las funciones auxiliares están anidadas dentro de la función estadística). Paralelo = "nieve" está diseñado principalmente para ser utilizado en máquinas Windows de múltiples núcleos donde paralelo = "multinúcleo" no está disponible.

Para la mayoría de los métodos de arranque, el remuestreo se realiza en el proceso maestro, pero no si simple = TRUE ni sim = "paramétrico". En esos casos (o en los que la estadística en sí usa números aleatorios), se necesita más cuidado si los resultados deben ser reproducibles. El remuestreo se realiza en los procesos de trabajo por censboot (sim = "wierd") y por la mayoría de los esquemas en tsboot (las excepciones son sim == "fixed" y sim == "geom" con el ran.gen predeterminado).

Cuando la generación de números aleatorios se realiza en los procesos de trabajo, el comportamiento predeterminado es que cada trabajador elige una semilla separada, de forma no reproducible. Sin embargo, con paralelo = "multinúcleo" o paralelo = "nieve" utilizando el grupo predeterminado, se utiliza un segundo enfoque si se ha seleccionado RNGkind ("L'Ecuyer-CMRG"). En ese enfoque, cada trabajador obtiene una subsecuencia diferente de la secuencia RNG basada en la semilla en el momento en que se genera el trabajador y, por lo tanto, los resultados serán reproducibles si ncpus no cambia, y para paralelo = "multinúcleo" si paralelo :: mc.reset .stream () se llama: vea los ejemplos de mclapply.

Tenga en cuenta que cargar el espacio de nombres paralelo puede cambiar la semilla aleatoria, por lo que para obtener la máxima reproducibilidad, debe hacerlo antes de llamar a esta función.

1.7. Referencias

Hay muchas referencias que explican el bootstrap y sus variaciones. Entre ellos están:

Booth, J.G., Hall, P. and Wood, A.T.A. (1993) Balanced importance resampling for the bootstrap. Annals of Statistics, 21, 286–298.

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

Davison, A.C., Hinkley, D.V. and Schechtman, E. (1986) Efficient bootstrap simulation. Biometrika, 73, 555–566.

Efron, B. and Tibshirani, R. (1993) An Introduction to the Bootstrap. Chapman & Hall.

Gleason, J.R. (1988) Algorithms for balanced bootstrap simulations. American Statistician, 42, 263–266.

Hall, P. (1989) Antithetic resampling for the bootstrap. Biometrika, 73, 713–724.

Hinkley, D.V. (1988) Bootstrap methods (with Discussion). Journal of the Royal Statistical Society, B, 50, 312–337, 355–370.

Hinkley, D.V. and Shi, S. (1989) Importance sampling and the nested bootstrap. Biometrika, 76, 435–446.

Johns M.V. (1988) Importance sampling for bootstrap confidence intervals. Journal of the American Statistical Association, 83, 709–714.

Noreen, E.W. (1989) Computer Intensive Methods for Testing Hypotheses. John Wiley & Sons.

1.8. Ejemplos

```
library(boot)
```

1.8.1. Ejemplo 1

```
# Usual bootstrap of the ratio of means using the city data
ratio <- function(d, w) sum(d$x * w)/sum(d$u * w)
boot(city, ratio, R = 999, stype = "w")
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = city, statistic = ratio, R = 999, stype = "w")
##
##
## Bootstrap Statistics :
##     original     bias    std. error
## t1* 1.520313 0.05041053   0.2169311
```

1.8.2. Ejemplo 2

```
# Stratified resampling for the difference of means.  In this
# example we will look at the difference of means between the final
# two series in the gravity data.
diff.means <- function(d, f)
{    n <- nrow(d)
     gp1 <- 1:table(as.numeric(d$series))[1]
```

```
     m1 <- sum(d[gp1,1] * f[gp1])/sum(f[gp1])
     m2 <- sum(d[-gp1,1] * f[-gp1])/sum(f[-gp1])
     ss1 <- sum(d[gp1,1]^2 * f[gp1]) - (m1 *  m1 * sum(f[gp1]))
     ss2 <- sum(d[-gp1,1]^2 * f[-gp1]) - (m2 *  m2 * sum(f[-gp1]))
     c(m1 - m2, (ss1 + ss2)/(sum(f) - 2))
}
grav1 <- gravity[as.numeric(gravity[,2]) >= 7,]
boot(grav1, diff.means, R = 999, stype = "f", strata = grav1[,2])
```

```
##
## STRATIFIED BOOTSTRAP
##
##
## Call:
## boot(data = grav1, statistic = diff.means, R = 999, stype = "f",
##     strata = grav1[, 2])
##
##
## Bootstrap Statistics :
##      original      bias    std. error
## t1* -2.846154  0.08385308    1.498399
## t2* 16.846154 -1.95112420    6.672061
```

1.8.3. Ejemplo 3

```
# In this example we show the use of boot in a prediction from
# regression based on the nuclear data.  This example is taken
# from Example 6.8 of Davison and Hinkley (1997).  Notice also
# that two extra arguments to 'statistic' are passed through boot.
nuke <- nuclear[, c(1, 2, 5, 7, 8, 10, 11)]
nuke.lm <- glm(log(cost) ~ date+log(cap)+ne+ct+log(cum.n)+pt, data = nuke)
nuke.diag <- glm.diag(nuke.lm)
nuke.res <- nuke.diag$res * nuke.diag$sd
nuke.res <- nuke.res - mean(nuke.res)

# We set up a new data frame with the data, the standardized
# residuals and the fitted values for use in the bootstrap.
nuke.data <- data.frame(nuke, resid = nuke.res, fit = fitted(nuke.lm))

# Now we want a prediction of plant number 32 but at date 73.00
new.data <- data.frame(cost = 1, date = 73.00, cap = 886, ne = 0,
                       ct = 0, cum.n = 11, pt = 1)
new.fit <- predict(nuke.lm, new.data)

nuke.fun <- function(dat, inds, i.pred, fit.pred, x.pred)
{
     lm.b <- glm(fit+resid[inds] ~ date+log(cap)+ne+ct+log(cum.n)+pt,
                 data = dat)
     pred.b <- predict(lm.b, x.pred)
     c(coef(lm.b), pred.b - (fit.pred + dat$resid[i.pred]))
}

nuke.boot <- boot(nuke.data, nuke.fun, R = 999, m = 1,
```

```
                  fit.pred = new.fit, x.pred = new.data)
# The bootstrap prediction squared error would then be found by
mean(nuke.boot$t[, 8]^2)
```

```
## [1] 0.08541601
```

```
# Basic bootstrap prediction limits would be
new.fit - sort(nuke.boot$t[, 8])[c(975, 25)]
```

```
## [1] 6.133496 7.271004
```

1.8.4. Ejemplo 4

```
# Finally a parametric bootstrap.  For this example we shall look
# at the air-conditioning data.  In this example our aim is to test
# the hypothesis that the true value of the index is 1 (i.e. that
# the data come from an exponential distribution) against the
# alternative that the data come from a gamma distribution with
# index not equal to 1.
air.fun <- function(data) {
     ybar <- mean(data$hours)
     para <- c(log(ybar), mean(log(data$hours)))
     ll <- function(k) {
          if (k <= 0) 1e200 else lgamma(k)-k*(log(k)-1-para[1]+para[2])
     }
     khat <- nlm(ll, ybar^2/var(data$hours))$estimate
     c(ybar, khat)
}

air.rg <- function(data, mle) {
    # Function to generate random exponential variates.
    # mle will contain the mean of the original data
    out <- data
    out$hours <- rexp(nrow(out), 1/mle)
    out
}

air.boot <- boot(aircondit, air.fun, R = 999, sim = "parametric",
                 ran.gen = air.rg, mle = mean(aircondit$hours))

# The bootstrap p-value can then be approximated by
sum(abs(air.boot$t[,2]-1) > abs(air.boot$t0[2]-1))/(1+air.boot$R)
```

```
## [1] 0.466
```

2. Intervalos de confianza no paramétricos bootstrap con boot.ci

2.1. Descripción

Esta función genera 5 tipos diferentes de intervalos de confianza no paramétricos de dos colas equidistantes. Estos son:

- la aproximación normal de primer orden,
- el intervalo de arranque básico,
- el intervalo de arranque estudentizado,
- el intervalo de percentil de arranque y
- el intervalo de percentil de arranque ajustado (BCa).

Se pueden generar todos o un subconjunto de estos intervalos.

2.2. Sintaxis

```
boot.ci(boot.out, conf = 0.95, type = "all",
        index = 1:min(2,length(boot.out$t0)), var.t0 = NULL,
        var.t = NULL, t0 = NULL, t = NULL, L = NULL,
        h = function(t) t, hdot = function(t) rep(1,length(t)),
        hinv = function(t) t, ...)
```

2.3. Argumentos

- `boot.out`: An object of class "boot" containing the output of a bootstrap calculation.
- `conf`:
  A scalar or vector containing the confidence level(s) of the required interval(s).
- `type`:
  A vector of character strings representing the type of intervals required. The value should be any subset of the values c("norm","basic", "stud", "perc", "bca") or simply "all" which will compute all five types of intervals.
- `index`:
  This should be a vector of length 1 or 2. The first element of index indicates the position of the variable of interest in boot.out$t0 and the relevant column in boot.out$t. The second element indicates the position of the variance of the variable of interest. If both var.t0 and var.t are supplied then the second element of index (if present) is ignored. The default is that the variable of interest is in position 1 and its variance is in position 2 (as long as there are 2 positions in boot.out$t0).
- `var.t0`: If supplied, a value to be used as an estimate of the variance of the statistic for the normal approximation and studentized intervals. If it is not supplied and length(index) is 2 then var.t0 defaults to boot.out$t0[index[2]] otherwise var.t0 is undefined. For studentized intervals var.t0 must be defined. For the normal approximation, if var.t0 is undefined it defaults to var(t). If a transformation is supplied through the argument h then var.t0 should be the variance of the untransformed statistic.
- `var.t`:
  This is a vector (of length `boot.out$R`) of variances of the bootstrap replicates of the variable of interest. It is used only for studentized intervals. If it is not supplied and length(index) is 2 then var.t defaults to `boot.out$t[,index[2]]`, otherwise its value is undefined which will cause an error for studentized intervals. If a transformation is supplied through the argument h then var.t should be the variance of the untransformed bootstrap statistics.
- `t0`: The observed value of the statistic of interest. The default value is `boot.out$t0[index[1]]`. Specification of t0 and t allows the user to get intervals for a transformed statistic which may not be in

the bootstrap output object. See the second example below. An alternative way of achieving this would be to supply the functions h, hdot, and hinv below.

- `t`:
  The bootstrap replicates of the statistic of interest. It must be a vector of length `boot.out$R`. It is an error to supply one of t0 or t but not the other. Also if studentized intervals are required and t0 and t are supplied then so should be var.t0 and var.t. The default value is `boot.out$t[,index]`.

- `L`:
  The empirical influence values of the statistic of interest for the observed data. These are used only for BCa intervals. If a transformation is supplied through the parameter h then L should be the influence values for t; the values for h(t) are derived from these and hdot within the function. If L is not supplied then the values are calculated using empinf if they are needed.

- `h`:
  A function defining a transformation. The intervals are calculated on the scale of h(t) and the inverse function hinv applied to the resulting intervals. It must be a function of one variable only and for a vector argument, it must return a vector of the same length, i.e. h(c(t1,t2,t3)) should return c(h(t1),h(t2),h(t3)). The default is the identity function.

- `hdot`: A function of one argument returning the derivative of h. It is a required argument if h is supplied and normal, studentized or BCa intervals are required. The function is used for approximating the variances of h(t0) and h(t) using the delta method, and also for finding the empirical influence values for BCa intervals. Like h it should be able to take a vector argument and return a vector of the same length. The default is the constant function 1.

- `hinv`: A function, like h, which returns the inverse of h. It is used to transform the intervals calculated on the scale of h(t) back to the original scale. The default is the identity function. If h is supplied but hinv is not, then the intervals returned will be on the transformed scale.

- `...`:
  Any extra arguments that `boot.out$statistic` is expecting. These arguments are needed only if BCa intervals are required and L is not supplied since in that case L is calculated through a call to empinf which calls `boot.out$statistic`.

2.4. Valores

Un objeto de tipo “bootci” el cual contiene los intervalos. Tiene las siguientes componentes:

- `R`:
  The number of bootstrap replicates on which the intervals were based.

- `t0`: The observed value of the statistic on the same scale as the intervals.

- `call`:
  The call to boot.ci which generated the object.

  It will also contain one or more of the following components depending on the value of type used in the call to bootci.

- `normal`: A matrix of intervals calculated using the normal approximation. It will have 3 columns, the first being the level and the other two being the upper and lower endpoints of the intervals.

- `basic`:
  The intervals calculated using the basic bootstrap method.

- `student`: The intervals calculated using the studentized bootstrap method.

- `percent`: The intervals calculated using the bootstrap percentile method.

- `bca`: The intervals calculated using the adjusted bootstrap percentile (BCa) method.

These latter four components will be matrices with 5 columns, the first column containing the level, the next two containing the indices of the order statistics used in the calculations and the final two the calculated endpoints themselves.

2.5. Ejemplos

2.5.1. Ejemplo 1

```
# confidence intervals for the city data
ratio <- function(d, w) sum(d$x * w)/sum(d$u * w)
city.boot <- boot(city, ratio, R = 999, stype = "w", sim = "ordinary")
boot.ci(city.boot, conf = c(0.90, 0.95),
        type = c("norm", "basic", "perc", "bca"))
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 999 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = city.boot, conf = c(0.9, 0.95), type = c("norm",
##     "basic", "perc", "bca"))
##
## Intervals :
## Level      Normal              Basic
## 90%   ( 1.098,  1.851 )   ( 1.056,  1.755 )
## 95%   ( 1.025,  1.923 )   ( 0.923,  1.795 )
##
## Level     Percentile            BCa
## 90%   ( 1.286,  1.985 )   ( 1.275,  1.964 )
## 95%   ( 1.246,  2.118 )   ( 1.241,  2.087 )
## Calculations and Intervals on Original Scale
```

2.5.2. Ejemplo 2

```
# studentized confidence interval for the two sample
# difference of means problem using the final two series
# of the gravity data.
diff.means <- function(d, f)
{    n <- nrow(d)
     gp1 <- 1:table(as.numeric(d$series))[1]
     m1 <- sum(d[gp1,1] * f[gp1])/sum(f[gp1])
     m2 <- sum(d[-gp1,1] * f[-gp1])/sum(f[-gp1])
     ss1 <- sum(d[gp1,1]^2 * f[gp1]) - (m1 *  m1 * sum(f[gp1]))
     ss2 <- sum(d[-gp1,1]^2 * f[-gp1]) - (m2 *  m2 * sum(f[-gp1]))
     c(m1 - m2, (ss1 + ss2)/(sum(f) - 2))
}
grav1 <- gravity[as.numeric(gravity[,2]) >= 7, ]
grav1.boot <- boot(grav1, diff.means, R = 999, stype = "f",
                   strata = grav1[ ,2])
boot.ci(grav1.boot, type = c("stud", "norm"))
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 999 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = grav1.boot, type = c("stud", "norm"))
```

```
##
## Intervals :
## Level      Normal            Studentized
## 95%   (-5.846,  0.240 )   (-7.152, -0.071 )
## Calculations and Intervals on Original Scale
```

2.5.3. Ejemplo 3

```
# Nonparametric confidence intervals for mean failure time
# of the air-conditioning data as in Example 5.4 of Davison
# and Hinkley (1997)
mean.fun <- function(d, i)
{    m <- mean(d$hours[i])
     n <- length(i)
     v <- (n-1)*var(d$hours[i])/n^2
     c(m, v)
}
air.boot <- boot(aircondit, mean.fun, R = 999)
boot.ci(air.boot, type = c("norm", "basic", "perc", "stud"))
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 999 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = air.boot, type = c("norm", "basic", "perc",
##     "stud"))
##
## Intervals :
## Level      Normal              Basic
## 95%   ( 37.2, 180.9 )   ( 34.7, 169.0 )
##
## Level    Studentized          Percentile
## 95%   ( 52.7, 298.6 )   ( 47.2, 181.5 )
## Calculations and Intervals on Original Scale
```

2.5.4. Ejemplo 4

```
# Now using the log transformation
# There are two ways of doing this and they both give the
# same intervals.

# Method 1
boot.ci(air.boot, type = c("norm", "basic", "perc", "stud"),
        h = log, hdot = function(x) 1/x)
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 999 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = air.boot, type = c("norm", "basic", "perc",
##     "stud"), h = log, hdot = function(x) 1/x)
##
## Intervals :
## Level      Normal              Basic
```

```
## 95%   ( 4.043,  5.467 )   ( 4.165,  5.512 )
##
## Level    Studentized          Percentile
## 95%   ( 4.024,  5.789 )   ( 3.854,  5.201 )
## Calculations and Intervals on  Transformed Scale
```

```
# Method 2
vt0 <- air.boot$t0[2]/air.boot$t0[1]^2
vt <- air.boot$t[, 2]/air.boot$t[ ,1]^2
boot.ci(air.boot, type = c("norm", "basic", "perc", "stud"),
        t0 = log(air.boot$t0[1]), t = log(air.boot$t[,1]),
        var.t0 = vt0, var.t = vt)
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 999 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = air.boot, type = c("norm", "basic", "perc",
##     "stud"), var.t0 = vt0, var.t = vt, t0 = log(air.boot$t0[1]),
##     t = log(air.boot$t[, 1]))
##
## Intervals :
## Level      Normal              Basic
## 95%   ( 4.072,  5.438 )   ( 4.165,  5.512 )
##
## Level    Studentized          Percentile
## 95%   ( 4.024,  5.789 )   ( 3.854,  5.201 )
## Calculations and Intervals on Original Scale
```