Bootstrapping

Jo Hardin

October 4, 2017

Why bootstrap?

Motivation: to estimate the variability of a statistic (*not* dependent on H_0 being true).

Example

- Hesketh and Everitt (2000) report on a study by Caplehorn and Bell (1991) that investigated the times that heroin addicts remained in a clinic for methadone maintenance treatment.
- The data include the amount of time that the subjects stayed in the facility until treatment was terminated (column 4).
- For about 37% of the subjects, the study ended while they were still the in clinic (status=0).
- Their survival time has been truncated. For this reason we might not want to estimate the mean survival time, but rather some other measure of typical survival time. Below we explore using the median as well as the 25% trimmed mean. (From ISCAM Chance & Rossman, Investigation 4.5.3)

Reading in the data

```
heroin <- read.delim("HEROIN.TXT")
names(heroin)
```

```
## [1] "id"     "clinic" "status" "time"   "prison" "dose"
```

```
head(heroin)
```

```
##   id clinic status time prison dose
## 1  1      1      1  428      0   50
## 2  2      1      1  275      1   55
## 3  3      1      1  262      0   55
## 4  4      1      1  183      0   30
## 5  5      1      1  259      1   65
## 6  6      1      1  714      0   55
```

Observed Test Statistic(s)

```
obs.stat<-heroin %>% summarize(medtime = median(time)) %>% pull()
obs.stat2<-heroin %>% summarize(tmeantime = mean(time, trim=0.25)) %>% pull()
obs.stat
```

```
## [1] 368
```

```
obs.stat2
```

```
## [1] 378
```

Bootstrapped data!

```
set.seed(4747)
heroin.rs<-heroin %>% sample_frac(size=1, replace=TRUE)

heroin.rs %>% summarize(medtime = median(time)) %>% pull()
```

```
## [1] 368
```

```
heroin.rs %>% summarize(tmeantime = mean(time, trim=0.25)) %>% pull()
```

```
## [1] 392
```

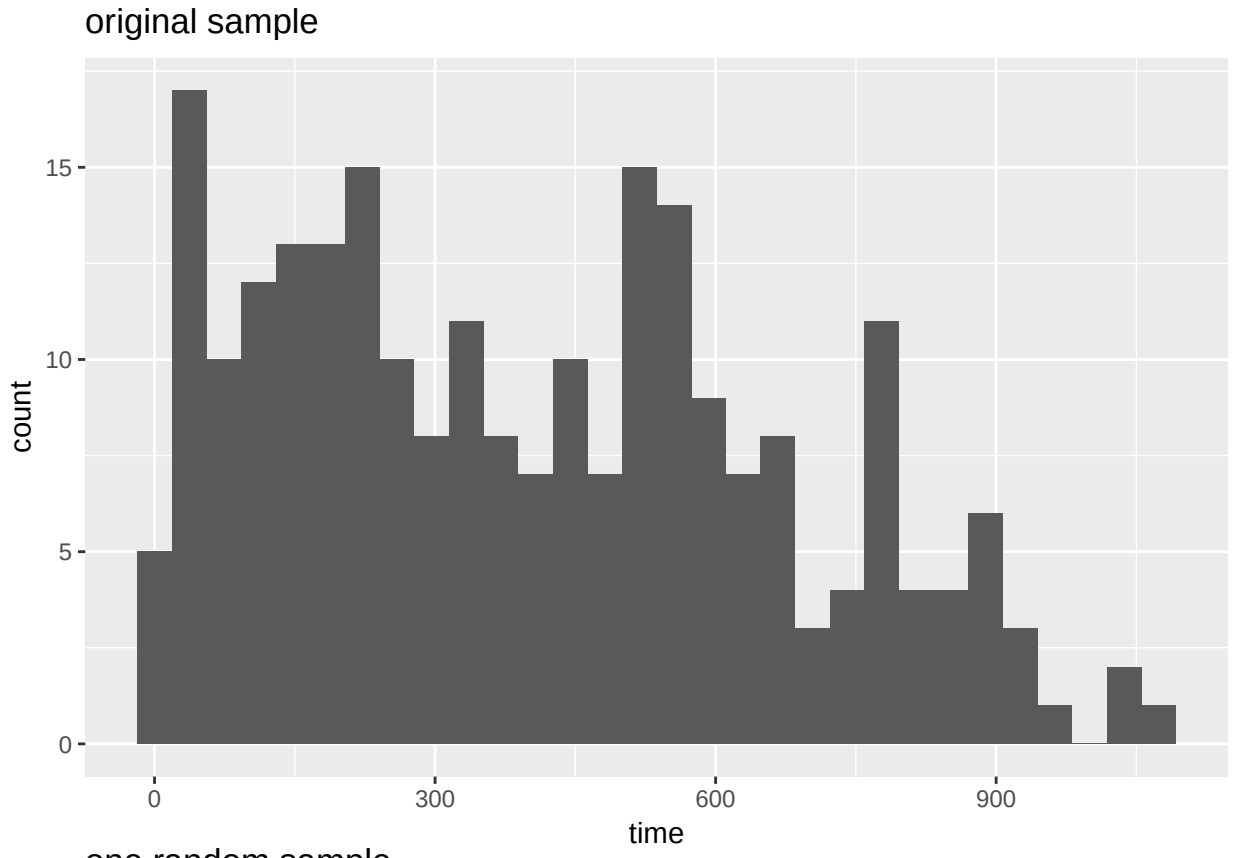
Need to bootstrap a lot of times. . .

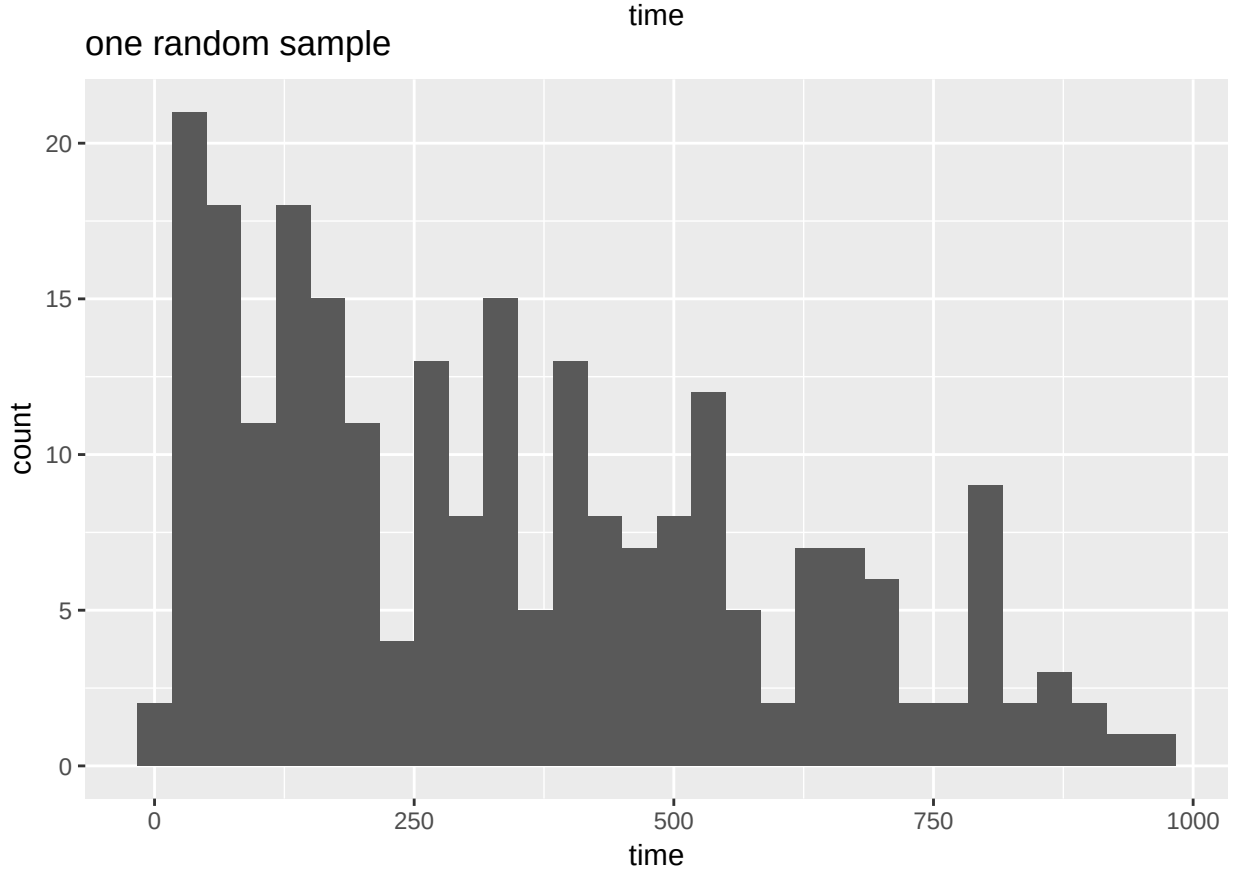
Below is the code showing how to bootstrap using for loops (nested to create the t multipliers needed for the BS-t intervals). However, the package and funciton `boot` will do the boostrapping for you.

```
test.stat<-c()
test.stat2<-c()
sd.test.stat<-c()
sd.test.stat2<-c()

reps1 <- 200
reps2 <- 25

set.seed(4747)
for(i in 1:reps1){
    heroin.rs<-heroin %>% sample_frac(size=1, replace=TRUE)
    test.stat<-c(test.stat,heroin.rs %>% summarize(medtime = median(time)) %>% pull())
    test.stat2<-c(test.stat2,heroin.rs %>% summarize(tmeantime = mean(time, trim=0.25)) %>% pull())

    test.stat.rs<-c()
    test.stat2.rs<-c()

    for(j in 1:reps2){
    heroin.rsrs<-heroin %>% sample_frac(size=1, replace=TRUE)
    test.stat.rs<-c(test.stat.rs,heroin.rsrs %>% summarize(medtime = median(time)) %>% pull())
    test.stat2.rs<-c(test.stat2.rs,heroin.rsrs %>% summarize(tmeantime = mean(time, trim=0.25)) %>% pul
    }
    sd.test.stat<-c(sd.test.stat,sd(test.stat.rs))
    sd.test.stat2<-c(sd.test.stat2,sd(test.stat2.rs))
}
```

What do the distributions look like?

The distributions of both the median and the trimmed mean are symmetric and bell-shaped. However, the trimmed mean has a more normal distribution (as evidenced by the points of the qq plot falling on the line y=x).

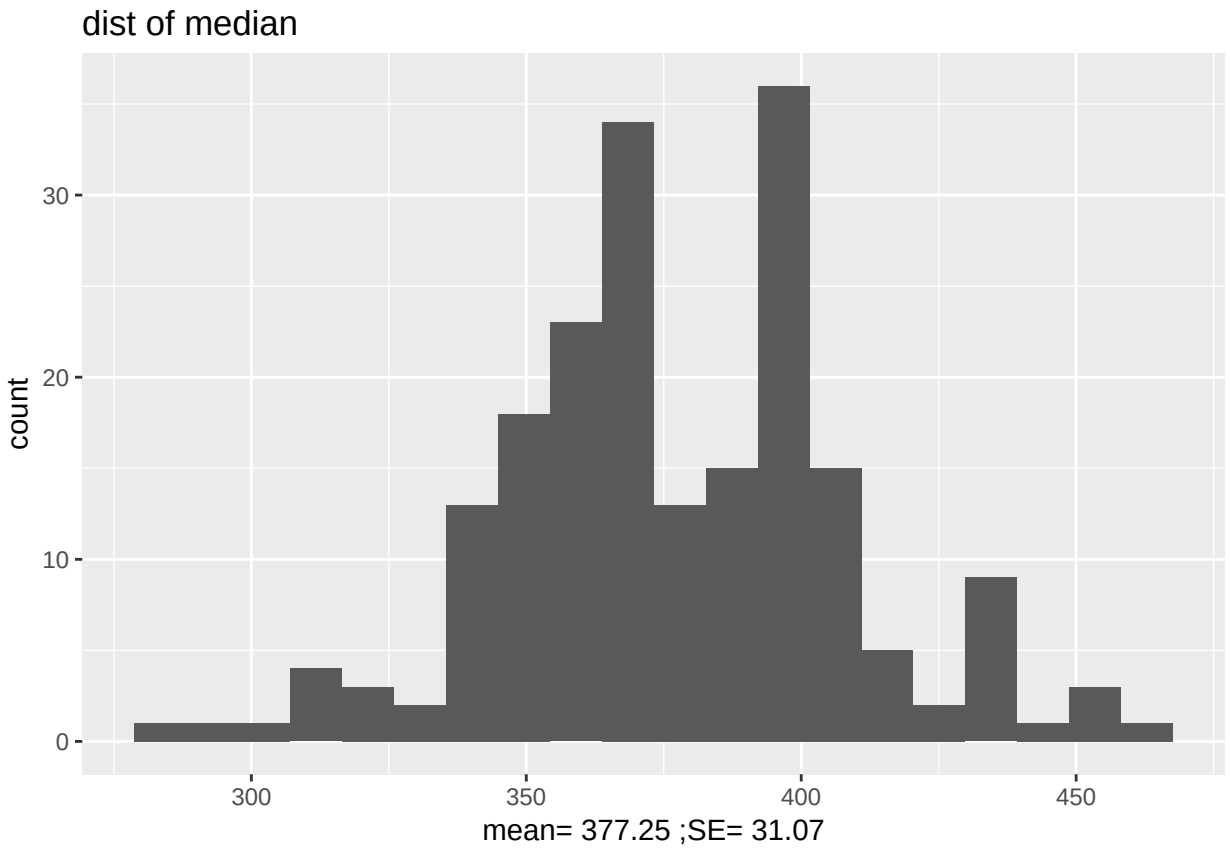

original sample
count
time
0
5
10
15
0
300
600
900


one random sample
count
time
0
5
10
15
20
0
250
500
750
1000

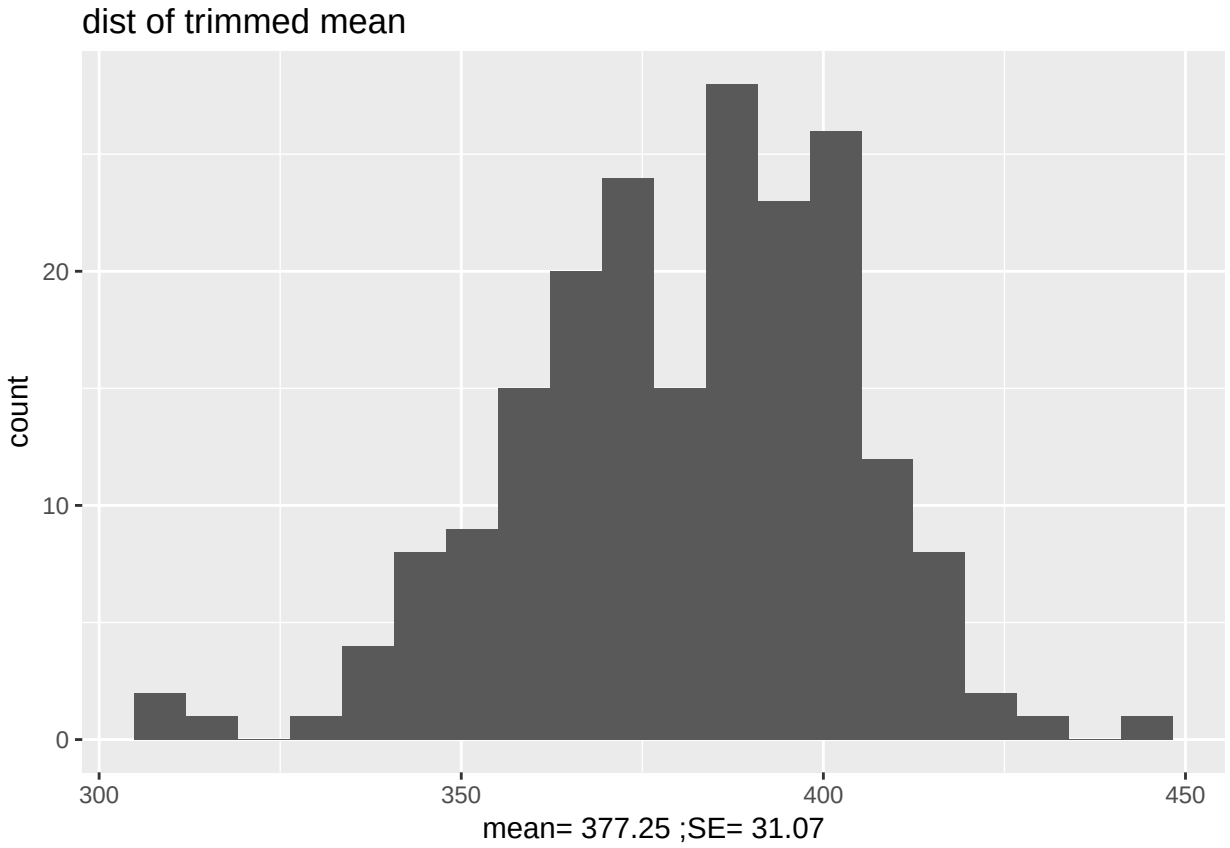
What do the distributions look like?

The distributions of both the median and the trimmed mean are symmetric and bell-shaped. However, the trimmed mean has a more normal distribution (as evidenced by the points of the qq plot falling on the line y=x).

```
bs.stats <- data.frame(test.stat, test.stat2)
ggplot(bs.stats, aes(x=test.stat)) + geom_histogram(bins=20) + ggtitle("dist of median") +  xlab(paste(
```




```
ggplot(bs.stats, aes(x=test.stat2)) + geom_histogram(bins=20) + ggtitle("dist of trimmed mean") +  xlab
```



OR using the built in functions

```
sampletmean <- function(x,d,trimperc){
  return(mean(x[d], trim=trimperc))
}
set.seed(4747)
bs.tmean.resamps <- boot(heroin[,4],sampletmean, reps1, trimperc=.25)
```

What does the boot output look like?

```
# bs.tmean.resamps <- boot(heroin[,4],sampletmean, reps1, trimperc=.25)
str(bs.tmean.resamps)
```

```
## List of 11
##  $ t0       : num 378
##  $ t        : num [1:200, 1] 393 379 377 381 379 ...
##  $ R        : num 200
##  $ data     : num [1:238] 428 275 262 183 259 714 438 796 892 393 ...
##  $ seed     : int [1:626] 403 624 -1645349161 -2081516244 1489809469 823736794 -755145325 950390200 -
##  $ statistic:function (x, d, trimperc)
##   ..- attr(*, "srcref")=Class 'srcref'  atomic [1:8] 1 16 3 1 16 1 1 3
##   .. .. ..- attr(*, "srcfile")=Classes 'srcfilecopy', 'srcfile' <environment: 0x7f8dba551f48>
##  $ sim      : chr "ordinary"
##  $ call     : language boot(data = heroin[, 4], statistic = sampletmean, R = reps1,      trimperc = 
```

```
##  $ stype    : chr "i"
##  $ strata   : num [1:238] 1 1 1 1 1 1 1 1 1 1 ...
##  $ weights  : num [1:238] 0.0042 0.0042 0.0042 0.0042 0.0042 ...
##  - attr(*, "class")= chr "boot"
```

```
samplemed <- function(x,d){
  return(median(x[d]))
}
set.seed(4747)
bs.med.resamps <- boot(heroin[,4],samplemed, reps1)
```

SE of median

Whew! They are very close (one using for loops, one using the boot function).

```
sd(test.stat)  # SE of median
```

```
## [1] 31.1
```

```
bs.med.resamps
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = heroin[, 4], statistic = samplemed, R = reps1)
##
##
## Bootstrap Statistics :
##     original  bias    std. error
## t1*      368     6.9          33
```

SE of trimmed mean

Whew! They are very close (one using for loops, one using the boot function).

```
sd(test.stat2)  # SE of trimmed mean
```

```
## [1] 22.7
```

```
bs.tmean.resamps
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = heroin[, 4], statistic = sampletmean, R = reps1,
##     trimperc = 0.25)
##
##
## Bootstrap Statistics :
##     original  bias    std. error
## t1*      378   -1.07          22
```

95% normal CI with BS SE

Without built in functions

```
obs.stat + qt(c(.025,.975),nrow(heroin)-1)*sd(test.stat)
```

```
## [1] 306 429
```

```
obs.stat2 + qt(c(.025,.975),nrow(heroin)-1)*sd(test.stat2)
```

```
## [1] 334 423
```

With built in functions

```
se.bs <- sd(bs.med.resamps$t)
se.bs2 <- sd(bs.tmean.resamps$t)

obs.stat + qt(c(0.025,.975), nrow(heroin) - 1)*se.bs
```

```
## [1] 302 433
```

```
obs.stat2 + qt(c(0.025,.975), nrow(heroin) - 1)*se.bs2
```

```
## [1] 335 422
```

95% Percentile CI

Without built in functions

```
quantile(test.stat, c(.025, .975))
```

```
##  2.5% 97.5%
##   315   439
```

```
quantile(test.stat2, c(.025, .975))
```

```
##  2.5% 97.5%
##   337   419
```

With built in functions

```
quantile(bs.med.resamps$t, c(.025, .975))
```

```
##  2.5% 97.5%
##   323   445
```

```
quantile(bs.tmean.resamps$t, c(.025, .975))
```

```
##  2.5% 97.5%
##   333   419
```

With built in functions more directly

```
boot.ci(bs.med.resamps, type="perc")
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 200 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = bs.med.resamps, type = "perc")
##
## Intervals :
## Level     Percentile
## 95%   (322, 453 )
## Calculations and Intervals on Original Scale
## Some percentile intervals may be unstable
```

```
boot.ci(bs.tmean.resamps, type="perc")
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 200 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = bs.tmean.resamps, type = "perc")
##
## Intervals :
## Level     Percentile
## 95%   (332, 422 )
## Calculations and Intervals on Original Scale
## Some percentile intervals may be unstable
```

95% Bootstrap-t CI

Without built in functions

Note that the t-value is needed (which requires a different SE for each bootstrap sample).

```
t.hat<-(test.stat - obs.stat)/sd.test.stat
t.hat2<-(test.stat2 - obs.stat2)/sd.test.stat2

t.hat.95 = quantile(t.hat, c(.025,.975))
t.hat2.95 = quantile(t.hat2, c(.025,.975))

obs.stat + t.hat.95*sd(test.stat)
```

```
##  2.5% 97.5%
##   314   439
```

```
obs.stat2 + t.hat2.95*sd(test.stat2)
```

```
##  2.5% 97.5%
##   333   420
```

With built in functions

Trimmed mean:

```
sampletmean2 <- function(x, d, R2, trimperc) {
   boot.samp = x[d]  # boostrapped sample
   m.bs = mean(boot.samp, trim=trimperc)  # bootstrapped mean
   v.bs = var(boot(boot.samp, sampletmean, R2, trim=trimperc)$t)
   return(c(m.bs, v.bs))  # boot expects the statistic to be the 1st and the var to be the 2nd
}
set.seed(4747)
bs.tmean.reresamps <- boot(heroin[,4], sampletmean2, R=reps1, R2=reps2, trimperc=.25)
```

Median:

```
samplemed2 <- function(x, d, R2) {
   boot.samp = x[d]  # boostrapped sample
   m.bs = median(boot.samp)  # bootstrapped mean
   v.bs = var(boot(boot.samp, samplemed, R2)$t)
   return(c(m.bs, v.bs))  # boot expects the statistic to be the 1st and the var to be the 2nd
}
set.seed(4747)
bs.med.reresamps <- boot(heroin[,4], samplemed2, R=reps1, R2=reps2)
```

The confidence intervals (BS-t intervals, called "studentized"):

```
boot.ci(bs.med.reresamps, type="stud")
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 200 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = bs.med.reresamps, type = "stud")
##
## Intervals :
## Level    Studentized
## 95%   (274, 419 )
## Calculations and Intervals on Original Scale
## Some studentized intervals may be unstable
```

```
boot.ci(bs.tmean.reresamps, type="stud")
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 200 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = bs.tmean.reresamps, type = "stud")
##
## Intervals :
## Level    Studentized
## 95%   (334, 428 )
## Calculations and Intervals on Original Scale
## Some studentized intervals may be unstable
```

95% BCa interval (not responsible for BCa)

With built in functions

```
boot.ci(bs.med.reresamps, type="bca")
boot.ci(bs.tmean.reresamps, type="bca")
```

Without built in functions

```
test.stat.jk<-c()
test.stat2.jk<-c()

set.seed(4747)
for(i in 1:length(heroin[,4])){

    test.stat.jk<-c(test.stat.jk,median(heroin[-i,4]))
     test.stat2.jk<-c(test.stat2.jk,mean(heroin[-i,4],trim=.25))
}

zo.hat<-qnorm(sum(test.stat<obs.stat)/reps1,0,1)
a.hat<- sum((mean(test.stat.jk) - test.stat.jk)^3)/
                (6*(sum((mean(test.stat.jk)-test.stat.jk)^2)^1.5))

zo.hat2<- qnorm(sum(test.stat2< obs.stat2)/reps1,0,1)
a.hat2<- sum((mean(test.stat2.jk) - test.stat2.jk)^3)/
                (6*(sum((mean(test.stat2.jk)-test.stat2.jk)^2)^1.5))

alpha1.bca<-pnorm(zo.hat + (zo.hat + qnorm(.975))/(1 - a.hat*(zo.hat + qnorm(.975))))
alpha2.bca<-pnorm(zo.hat + (zo.hat + qnorm(.025))/(1 - a.hat*(zo.hat + qnorm(.025))))


alpha1.bca2<-pnorm(zo.hat2 + (zo.hat2 + qnorm(.975))/(1 - a.hat2*(zo.hat2 + qnorm(.975))))
alpha2.bca2<-pnorm(zo.hat2 + (zo.hat2 + qnorm(.025))/(1 - a.hat2*(zo.hat2 + qnorm(.025))))


c(sort(test.stat)[ceiling(reps1*alpha2.bca)],sort(test.stat)[ceiling(reps1*alpha1.bca)])
c(sort(test.stat2)[ceiling(reps1*alpha2.bca2)],sort(test.stat2)[ceiling(reps1*alpha1.bca2)])
```

Comparison of intervals

The first three columns correspond to the CIs for the true median of the survival times. The second three columns correspond to the CIs for the true trimmed mean of the survival times.

CI	lower	observed	upper	lower	observed	upper
Percentile	321.00	367.50	452.00	339.38	378.30	423.46
CI w BS SE	306.33	367.50	428.67	335.21	378.30	421.39
BS-t	294.98	367.50	418.00	334.28	378.30	418.09
BCa	317.00	367.50	444.00	338.29	378.30	422.43