Introduction to the Bootstrap and Robust Statistics

PSY711/712

Winter Term 2009

1 t-based Confidence Intervals

This document shows how to calculate confidence intervals for a sample mean using the t statistic in R.

1.1 using a `for` loop

Here is some R code that uses a `for` loop to generate a confidence interval using the percentile-t bootstrap method.

The first thing we need to do is figure out how to extract a t value from the output of `t.test`:

```
> n <- 20
> mu <- 50
> sigma <- 10
> set.seed(321)
> mySample <- rnorm(n=n,mean=mu,sd=sigma)
> t.test.result <- t.test(mySample)
> names(t.test.result)

[1] "statistic"   "parameter"   "p.value"     "conf.int"    "estimate"
[6] "null.value"  "alternative" "method"      "data.name"

> (the.t.value <- t.test.result$statistic)

      t
27.5021
```

Now that we can extract the t statistic, we can do the bootstrap. In the following code, notice that the call to `t.test` inside the `for` loop uses the sample mean, m, as an estimate of the population mean, μ. Although using m in this way may seem puzzling, it is consistent with the general bootstrap technique of using the sample as a stand-in for the actual population of scores.

```
> # need sample mean and sd:
> m <- mean(mySample) # need mean of original sample
> s <- sd(mySample)
> # now do the bootstrap:
> R <- 1999
> boot.t.value <- rep(0,R) # array for holding bootstrapped t's
> for (k in 1:R){
+         boot.sample <- sample(mySample,replace=T) # bootstrapped sample
+         # do t test:
+         boot.t.test <- t.test(boot.sample,mu=m) # notice that mu is sample mean!
+         boot.t.value[k] <- boot.t.test$statistic # extract t value
```

```
+ }
> # confidence interval:
> alpha <- .05;
> critical.t.low <- quantile(boot.t.value,alpha/2)
> critical.t.high <- quantile(boot.t.value,1-alpha/2)
> ci.95.bootstrap <- c(m  - critical.t.high*s/sqrt(n), m - critical.t.low*s/sqrt(n))
> names(ci.95.bootstrap) <- c("2.5%","97.5%")
> ci.95.bootstrap

    2.5%    97.5%
49.17219 57.62760
```

For comparison, here is the confidence interval calculated using classical methods:

```
> t.low <- qt(p=alpha/2,df=n-1)
> t.high <- qt(p=1-alpha/2,df=n-1)
> ci.95 <- c(m  - t.high*s/sqrt(n), m - t.low*s/sqrt(n))
> names(ci.95) <- c("2.5%","97.5%")
> ci.95

    2.5%    97.5%
48.91693 56.97581

> # alternative method:
> t.test(mySample)$conf.int[1:2]

[1] 48.91693 56.97581
```

The fact that the two intervals are so similar inspires some confidence in the bootstrap.

1.2 using `boot` and `boot.ci`

This section describe how to use the functions `boot` and `boot.ci` in the `boot` package to calculate percentile-t confidence intervals.

To use `boot` we need to write a function that calculates the statistic of interest – in this case the mean – on each bootstrapped sample. To calculate percentile-t confidence interval, we have to construct our function so that it also returns the variance of the estimated statistic. The function `boot.stat` fulfills these requirements:

```
> boot.mean <- function(theData,ids){
+         boot.sample <- theData[ids] # bootstrapped sample
+         m <- mean(boot.sample) # bootstrapped mean (the statistic of interest)
+         n <- length(boot.sample)
+         s <- sd(boot.sample) # pop sd estimated from bootstrapped sample
+         sem <- s/sqrt(n) # standard error of the bootstrapped mean
+         m.var <- sem^2 # IMPORTANT: boot.ci requires the estimated VARIANCE of the statistic
+         results <- c(m,m.var) # boot expects the statistic to be 1st and variance to be 2nd
+         return(results)
+ }
```

Notice that `boot.mean` returns *two* values: the mean of the bootstrapped sample and the *variance* of the sample mean. Also note that the order of the variables matters: `boot` expects the statistic to be given first, and the variance of the statistic second.

Next, we use `boot` to do the bootstrap and `boot.ci` to calculate the confidence interval. The percentile-t confidence interval is specified as `type="stud"` for *studentized* confidence interval.

```
> library(boot) # load package
> R <-1999
> boot.results <- boot(mySample,statistic=boot.mean,R=R)
> boot.ci(boot.results,type="stud") # studentized, or stud, equals percentile-t

BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1999 bootstrap replicates

CALL :
boot.ci(boot.out = boot.results, type = "stud")

Intervals :
Level    Studentized
95%   (49.39, 57.53 )
Calculations and Intervals on Original Scale
```

Not surprisingly, the confidence interval is similar to the one calculated with the classical method.

1.3 iterative bootstraps

Using a bootstrap does not eliminate the problems caused by outliers. Consider, for example, the confidence interval of the mean for the following data:

```
> set.seed(32)
> mySample <- rnorm(n=20)
> b <- boot(mySample,boot.mean,R=1999)
> boot.ci(b,type="stud")

BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1999 bootstrap replicates

CALL :
boot.ci(boot.out = b, type = "stud")

Intervals :
Level    Studentized
95%   (-0.0388,  0.4724 )
Calculations and Intervals on Original Scale

> mySample[20] <- 1000
> b <- boot(mySample,boot.mean,R=1999)
> boot.ci(b,type="stud")

BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1999 bootstrap replicates

CALL :
boot.ci(boot.out = b, type = "stud")

Intervals :
Level    Studentized
95%   (  -10.91, 28982.05 )
Calculations and Intervals on Original Scale
```

Introducing a single outlier results in a *dramatic* increase in the confidence interval. What is needed here, obviously, is a more robust estimate of central tendency. One possibility is the sample median: Why not alter `boot.mean` so that it calculates the median and estimates the variance of the sample median? Well, one problem is that it is not obvious how to calculate the variance of a sample median[1]. One way to get around this problem is to do a bootstrap *on the bootstrapped sample*. The following code shows how to do this bootstrap-within-a-bootstrap.

```
> boot.median <- function(theData,ids){
+         boot.sample <- theData[ids] # bootstrapped sample
+         m <- median(boot.sample) # bootstrapped median
+         n <- length(boot.sample)
+         R <- 199
+         # for loop is TOO SLOW:
+         # reboot.medians <- rep(0,R)
+         # for(k in 1:R){
+         #         reboot.sample <- sample(boot.sample,replace=T)
+         #         reboot.medians[k] <- median(reboot.sample)
+         # }
+         # the following three lines are MUCH faster than the for loop:
+         # create n x R matrix of resampled values:
+         reboot.samples <- matrix(data=sample(boot.sample,size=R*n,replace=T),nrow=n)
+         # calculate mean of each n=20 column
+         reboot.medians <- apply(reboot.samples,MARGIN=2,median)
+         # calculate variance of medians:
+         m.var <- var(reboot.medians)
+         #
+         # return results:
+         results <- c(m,m.var)
+         return(results)
+ }
> b <- boot(mySample,boot.median,R=1999)
> boot.ci(b,type="stud")

BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1999 bootstrap replicates

CALL :
boot.ci(boot.out = b, type = "stud")

Intervals :
Level    Studentized
95%   (-0.0217,  0.6303 )
Calculations and Intervals on Original Scale
```

Even without the `for` loop, the bootstrap takes about 1 minute to run on my computer (a MacBook Pro running OS X 10.5.5 with a 2.53 GHz Intel Core 2 Duo CPU). As expected, the confidence interval of the median is much narrower than the interval for the mean.

[1]There are, in fact, equations that can be used to estimate the standard error, and hence variance, of a sample median. However, for the purpose of this exercise we will ignore those equations.

1.4 limitations of t-based confidence intervals

There are many cases where our data consist of numbers that are ≥ 0. For instance, we may be measuring the number of offspring in different litters; the number of defects in silicon chips; the interval between the eruptions of a geyser; etc. In such cases, the sample mean *must* be ≥ 0. Notice, however, that confidence intervals calculated using t do not place any restriction on the lower value. In other words, a confidence interval based on t may include negative numbers even in cases where it is impossible to obtain such values. In other contexts, confidence intervals may exceed a maximum possible value for our measurements.

The following R code illustrates this problem with the classical method. In this example, we construct a data vector that must contain non-negative values:

```
> theData <- c(0,0,1,1,2,3,15,35)
> m <- mean(theData)
> s <- sd(theData)
> n <- length(theData)
> alpha <- .05
> t.low <- qt(p=alpha/2,df=n-1)
> t.high <- qt(p=1-alpha/2,df=n-1)
> ci.95 <- c(m  - t.high*s/sqrt(n), m - t.low*s/sqrt(n))
> names(ci.95) <- c("2.5%","97.5%")
> ci.95

     2.5%     97.5%
-3.157305 17.407305
```

A similar problem can occur with the percentile-t bootstrap

```
> set.seed(1)
> b <- boot(theData,boot.mean,R=999)
> boot.ci(b,type="stud")

BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 999 bootstrap replicates

CALL :
boot.ci(boot.out = b, type = "stud")

Intervals :
Level    Studentized
95%   (-0.367, 94.388 )
Calculations and Intervals on Original Scale
```

although, because of the random nature of the bootstrap, another bootstrap on the same data may yield a plausible confidence interval:

```
> set.seed(121)
> b <- boot(theData,boot.mean,R=999)
> boot.ci(b,type="stud")

BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 999 bootstrap replicates

CALL :
```

```
boot.ci(boot.out = b, type = "stud")

Intervals :
Level    Studentized
95%   ( 0.156, 84.319 )
Calculations and Intervals on Original Scale
```

Clearly, t-based methods can yield implausible confidence intervals. The problem here is that the requirement that the data be ≥ 0 is a gross violation of the normality assumption that underlies the use of t to construct a confidence interval. Interestingly, confidence intervals based on the percentile bootstrap do not suffer from this shortcoming. The following code re-creates the bootstrap that created the implausible studentized confidence interval and uses it to calculate the percentile and BCa confidence intervals:

```
> set.seed(1)
> b <- boot(theData,boot.mean,R=999)
> boot.ci(b,type=c("stud","perc","bca"))

BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 999 bootstrap replicates

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

Intervals :
Level    Studentized          Percentile            BCa
95%   (-0.367, 94.388 )   ( 0.875, 16.250 )   ( 1.375, 19.500 )
Calculations and Intervals on Original Scale
Some BCa intervals may be unstable
```

Unlike the studentized interval, the percentile and BCa intervals are plausible, although `boot.ci` returns a warning message suggesting that these intervals may not be reliable (probably because R is too small).