

Bootstrap Exercise

Some nonparametric bootstrap confidence interval techniques are sufficiently easy to implement that you should consider writing your own. Working with the breast cancer data, we can construct 95% confidence intervals for the 25% trimmed mean using both the percentile method and bias-corrected method. Copy the R routine `npcitrim` into R. The `npcitrim` function has default settings of `B=1000` bootstrap samples, `trim=.25`, and `alpha=.05` confidence level. A typical call would be (note that I have supplied the default values for the arguments here—that is typically unnecessary):

```
npcitrim(serum,trim=.25,B=1000,alpha=.05)
```

Run the subroutine, then construct the usual normal theory confidence interval for a sample mean, and compare your results. Modify the program to output the bootstrap estimates of mean and standard deviation and construct a normal bootstrap confidence interval:

$$\tilde{\theta}_n^*(\cdot) \pm z_\alpha s_B$$

Compare results again. This is a situation in which normal theory intervals behave reasonably well since the outlier problem and skewness are relatively modest.

R routines `boot` and `boot.ci`

Load the `boot` library from R. The R home webpage has other interesting bootstrap libraries to download. The `boot.ci` command operates on the output object from `boot`. We discussed `boot` in class and its cryptic use of a second argument (either a subscript, weight, or frequency) when calling the function that operates on each bootstrap sample. The second argument appears to control the sort of with replacement sampling that can take place, though its use often appears gratuitous in practice.

Let us revisit the breast cancer data using `boot` and `boot.ci`. Run the R script for these two commands. I specified the second argument to `trim` to be an index, though that is the default. Note that additional argument can be included in the call to `trim`; their values simply have to be specified as the final arguments in `boot`.

The percentile-t method requires a variance function in order to operate properly; the variance can be passed as the second output object in the bootstrap function. In cases such as the trimmed mean, there is no obvious functional form for the variance. If we were simply bootstrapping the mean, we could replace the final statement in `trim` with, e.g.,

```
return(mean(x[d]), (n-1)*var(x[d])/n^2).
```

This statement would automatically be accessed by `boot` if we wanted a percentile-t interval.

We can use `boot` to handle transformations of variables as well. Consider the law school data. We know that a better normal theory confidence interval could be produced if we used the \tanh^{-1} transformation on the correlation. Remember that this transformation will have relatively little effect on the performance of the percentile confidence interval though. There are a couple ways to obtain confidence intervals for transformed statistics in `boot.ci`. We can specify the transformation and its first derivative as the argument `h` and `hdot`. The first derivative is needed to compute the usual delta method estimator for the transformed estimator. If we want to back-transform the confidence interval endpoints, we also supply the `hinv` function. For a confidence interval for the sample correlation coefficient, this is relatively straightforward, since R has built-in \tanh and \tanh^{-1} functions.

Run the script to calculate confidence intervals for $\tanh^{-1}(\rho)$. If you would like a percentile-t estimate, you would need to modify the last statement in `corrlsat` to add the usual variance estimate of the correlation coefficient. You also need to specify the first derivative (argument `hdot`) of the transformation \tanh^{-1} in order to estimate the variance for the transformation using the delta method.

Run the modified script again, and you should obtain a percentile-t confidence interval for $\tanh^{-1}(\rho)$. These intervals often perform poorly.

Finally, provide the argument `hinv` to obtain appropriate confidence intervals in the original metric. This approach, using a normalizing transformation and then back-transforming endpoints, should improve the performance of the percentile-t, normal and basic methods.

As an alternate approach (not explored here), we can compute variance estimates by applying the delta method to the output from `boot`. The variable `t0` contains the sample estimate of r and the sample variance estimate of r (if calculated), while the variable `t` contains the B bootstrap estimates of r , and its variance estimate. These can be manipulated to generate variance estimates

for the transformed variable. In our case, however, we know the variance of $\tanh^{-1}(r)$ is equal to $1/(n-3)$, so that approach is not very instructive in this case.

Loess simulation

There is a bootstrap package available that conducts loess simulations. In our case, we will bootstrap by hand. Using the `cars` data set, we plot stopping distance as a function of speed. After fitting a loess curve, we save the fits, and the residuals. In our bootstrap routine `lboot`, we bootstrap the residuals, add them to the fitted values, fit a new loess curve, save it, and overlay it on our original data set. I would like you to run this exercise using the defaults (it's all in the script).

With the saved fitted values, we can construct empirical pointwise confidence intervals for each value of `speed`. It's better to do this with a larger bootstrap sample. Compute the confidence levels using the script and comment on your results.