Bootstrap Methods in R

Ella Kaye Xenia Miscouridou

October 21, 2015

Abstract

We describe a package `Bootstrap` in R which implements a variety of bootstrap tools: bootstrap, Bayesian bootstrap and bag of little bootstraps.

The package is available from `https://github.com/EllaKaye/Module1`.

1 Introduction

The bootstrap (Efron 1979), is a statistical tool which gives measures of accuracy of estimators, and can therefore be used draw inference about the parameters of the sampling distribution. It is a powerful and widely applicable tool. The bootstrap is at the interface between statistical inference and computation, and it was developed at a time when advances in computing technology allowed this computationally intensive method to be used in practice. Shortly after, the Bayesian analogue of the bootstrap was introduced (Rubin 1981). More recently, in this era of 'big data', statistical methodology is again being developed alongside developments in computing technology. Parallel and distributed computing architectures now make it possible to extend the bootstrap so that it can be applied to massive datasets, the so-called bag of little bootstraps (Kleiner et al. 2014).

We have developed a package, `Bootstrap`, with functions to implement the bootstrap, Bayesian bootstrap and bag of little bootstraps. The package can be obtained from GitHub.

```r
devtools::install_github("EllaKaye/Module1")
library(Bootstrap)
```

2 The Bootstrap

Suppose we observe a sample of n iid realisations $x_1, \ldots, x_n \sim P$, for some probability measure P. Let θ be a parameter of the distribution, and $\hat{\theta}_n$ be an estimator of θ. The goal of the bootstrap is to obtain an assessment, ξ, of the quality of the estimator. For example, θ_n could be the median, and ξ the standard error. To obtain a bootstrap estimate, we procede as follows:

1. Repeatedly (B times) sample n points with replacement from the original dataset, giving bootstrap replications (resamples) $(x_1^{*(i)}, \ldots, x_n^{*(i)}), i = 1, \ldots, B$

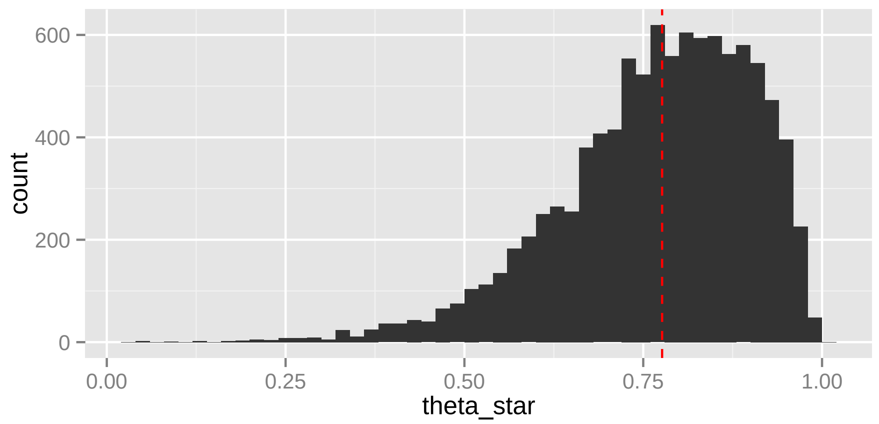
2. Compute $\hat{\theta}_n^{*(i)}$ on each of the B resamples.

3. Compute $\xi^* = \xi(\hat{\theta}_n^{*(1)}, \ldots, \hat{\theta}_n^{*(B)})$ as our estimate of ξ.

The `Bootstrap` library contains a function, `bootstrap`, which automates the above procedure. It takes as inputs a data set (which can be a vector, matrix or dataframe), a function, FUN, which is used to obtain the statistic of interest, and the number, B, of bootstrap resamples required. Note that the data must be passed to FUN in one object.

We now show how to use this function to replicate the example on page 37-38 of Efron and Gong (1983), where we are interested in obtaining the bootstrap estimate of the standard error of the Pearson correlation coefficient. The `Bootstrap` library contains a function, `cor_df`, which calculates the Pearson correlation coefficient between the first and second columns of a matrix or dataframe.

```r
law_school <- data.frame(
  LSAT=c(576,635,558,578,666,580,555,
         661,651,605,653,575,545,572,594),
  GPA=c(3.39,3.30,2.81,3.03,3.44,3.07,3.00,
        3.43,3.36,3.13,3.12,2.74,2.76,2.88,2.96))

set.seed(1)
bootstrap_law <- bootstrap(law_school, cor_df, B=10000)
bootstrap_law$se

## [1] 0.1347413

library(ggplot2)
bootstrap_law_df <- data.frame(
  theta_star = bootstrap_law$T_boot)
p <- qplot(theta_star, data=bootstrap_law_df, geom="histogram",
      binwidth=0.02)
p + geom_vline(xintercept=cor_df(law_school),
               colour="red", linetype="dashed")
```



When comparing the `bootstrap` function to the `boot` function in the package `boot`, we obtain the same estimate of the standard error. Both run in under 0.2 seconds. The `boot` function is negligibly faster, but we expect that that time difference may be increased for larger datasets.

3 Bayesian Bootstrap

Rubin (1981) introduced the Bayesian bootstrap (BB) as the natural Bayesian analogue of the bootstrap. Each BB replication generates a posterior probability for each x_i, which is centered at $1/n$, but has variability. To obtain a BB replication, first generate a set of weights by drawing $(n-1)$ uniform(0,1) random variates, $u_1, \dots, u_(n-1)$, ordering them and calculating the gaps $g_t = u_{(t)} - u_{(t-1)}, t = 1, \dots, n$, where $u_{(0)} = 0$ and $u_{(n)} = 1$. These gaps, $g = (g_1, \dots, g_n)$ form the weights to attach to the data values in that replication. Considering all the BB replications gives the BB distribution of X, and thus of any parameter of this distibution.

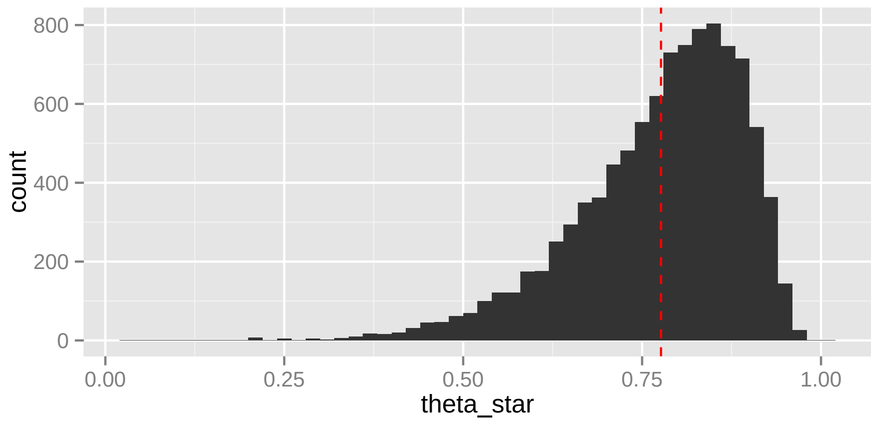
Our function `BB` gives a BB distribution. It takes as its inputs a dataset (as a vector, matrix or dataframe), a function for calculating the statistic $\hat{\theta}$ (this function must take two arguments - one for data and one for weights), and B, the number of replications required. It returns a list, the first item of which is the standard error of the replicates. The second item is a dataframe with $\theta_1^*, \dots, \theta_B^*$.

The following code gives the BB distribution when the parameter of interest is the mean. Here the function weighted.mean is in the R base package.

```
X <- rnorm(100, 4, 2)
BB_mean <- BB(X, weighted.mean, B=10000)
BB_mean$se

## [1] 0.1955357
```

It may be necessary to define your own function for the statistic of interest that can take weights as an argument. By way of example, we show the BB equivalent of the correlation example from the previous section. The function `cor` in the R base package cannot take weights; the fuction `weighted.cor` in our `Bootstrap` package does.

```
set.seed(1)
BB_law <- BB(law_school, weighted.cor, B=10000)
BB_law$se

## [1] 0.1163494

q <- qplot(T_boot, data=BB_law$replicates, geom="histogram",
      binwidth=0.02) + xlab("theta_star")
q + geom_vline(xintercept=cor_df(law_school),
               colour="red", linetype="dashed")
```



In the Bayesian bootstrap, the distribution is tighter around the original sample correlation than in the bootstrap case. In general, Rubin (1981) points out the inferential similarities between these two methods. As a result, whenever the applicability of the Bayesian bootstrap is called into question, we should also be weary of using the standard bootstrap. Typically, the standard bootstrap is seen as widely applicable, because it is completely non-parametric and does not seem to involve any model assumptions. However, both the bootstrap and Bayesian bootstrap operate under the assumption that all possible values of X have been observed. In any application, the researcher will need to ask themselves whether such an assumption is reasonable. In the Bayesian bootstrap, by using a draw from the uniform Dirichlet distribution for the weights, we are essentially placing a discrete improper prior on the data, which may not be reasonable if the underlying distibution is continuous.

4 Bag of Little Bootstraps

The original bootstrap arose around the time when increases in computing power allowed the development of statistical tools that had previously been too computationally expensive. In recent years, there has been an influx of 'big data', alongside the development of parallel computing architectures. Kleiner et al. (2014) have developed a scalable bootstrap for massive data, known as the Bag of Little Bootstraps (BLB). With massive datasets, the bootstrap's need for recomputation on resamples of the same size as the original dataset is problematic. Rather than obtain bootstrap samples from the whole dataset, the BLB breaks down the process as follows:

1. Repeatedly (s times) subsample $b(n) < n$ points *without replacement* from the original dataset of size n.
2. For each of the s subsamples, do the following:
   (a) Repeatedly (r times) resample n point *with replacement* from the subsample.
   (b) Compute $\hat{\theta}_n^*$ on each resample.
   (c) Compute an estimate of ξ based on these r realisations of $\hat{\theta}_n^*$.
3. We now have one estimate of ξ per subsample. Output their average as the final estimate of ξ for $\hat{\theta}_n$.

Kleiner et al. (2014) recommend taking $b(n) = n^\gamma$, where $\gamma \in [0.5, 1]$. This procedure dramatically reduces the size of each resample. For example, if $n =$ 1 million and $\gamma = 0.6$, the size of the original dataset may be around 1TB, with a bootstrap resample typically occupying approximately 632GB, and a BLB subsample or resample occupying just 4GB.

The Bootstrap package contains three functions which implement BLB in different cases, `BLB.1d, BLB.multi` and `BLB.adapt`. The function `BLB.1d` implements the simplest version of BLB. It takes as input: a 1-dimensional dataset in a vector; γ, which controls the value of b; a function FUN, which computes the parameter estimate, $\hat{\theta}_n$. It also takes as arguments s and r, which default to 20 and 100 respectively (Kleiner et al. (2014) demonstates that these values are likely as large as they'll need to be to obtain convergence). The function returns a BLB estimate of ξ, which is set as the standard error. When $n = 500,000$, the function takes under two minutes to execute (although we compute with a smaller sample here).

```
X <- rnorm(5000)
BLB.1d(X, mean, gamma=0.5)

## [1] 0.01355715
```

In their paper, Kleiner et al. (2014) conduct a simulation study. They generate data from the true underlying distribution, a linear model $Y_i = \tilde{X}_i^\top \mathbf{1}_d + \epsilon_i$, with iid $\tilde{X}_i \sim MVN(0, \mathbb{I}_d)$ and $\epsilon_i \sim N(0, 10)$. The estimator $\hat{\theta}_n$ consists of a linear least squares regression coefficients, with a small L_2 penalty of $\lambda = 10^{-5}$. To replicate their results, we created the function `BLB.multi`, which accepts as input an $n \times (d+1)$ dataframe or matrix, where each row is an observation, combining the d-dimensional x observation and, in the right-most column, the response, y. The tuning parameters for BLB, γ, r and s, can all be set, as can the tuning parameter λ for the penalised regression, and α, which controls the width of the confidence intervals. The function returns a list containing the BLB estimate of three quality measures, ξ^*: the confidence intervals for each dimension of the estimator, a vector with the width of those intervals, and the standard error of the estimator. As in the paper, we tested the function with $n = 20,000$, though we reduced the dimension d from 100 to 10. We chose $\gamma = 0.7$, kept $r = 100$, and set $s = 15$ (which is ample for convergence).

```r
# generate the data
n=20000
d=10
library(MASS)
set.seed(1)
X <- mvrnorm(n, mu=rep(0,d), Sigma=diag(d))
epsilon <- rnorm(n, mean=0, sd=sqrt(10))
t_theta <- as.matrix(rep(1,d))
Y <- X %*% t_theta + epsilon
data <- cbind(X,Y)

# apply the function and look at the results
system.time(
  BLB.multi_out <- BLB.multi(data, gamma=0.7, s=15, r=100)
)

##    user  system elapsed
##  66.092  14.649  80.791

BLB.multi_out$CI_width

##  [1] 0.08799343 0.08705251 0.08935000 0.08579360 0.08708979 0.09030666
##  [7] 0.08963174 0.08868356 0.08819788 0.08922422
```

As Kleiner et al. (2014) point out, when $n = 20,000$ the true average (across dimensions) marginal confidence interval width for the estimated parameter vector is approximately 0.1, so our function gives the expected results.

4.1 Adaptive tuning parameter selection

In `BLB.multi`, we need to specify values for s, the number of subsamples and r, the number of bootstrap replicates per subsample. As Kleiner et al. (2014) discuss, in practice fairly modest values for these tuning parameters often suffice for convergence, though they will vary depending on the value of γ, as well as on the parameter θ being estimated and the choice of quality assessment ξ. We wish to use the smallest values of r and s which have sufficiently good statistical properties. In order to minimise the computational cost, we can select r and s adaptively. Kleiner et al. (2014) suggest the following scheme. Suppose we are currenlty processing subsample j. We adaptively select r by repeatedly produce resamples, computing the parameter $\theta_n^\star$ and evaluating $\xi_n^\star$, until convergence. Convergence is said to have occurred when $\xi_n^\star$ no longer changes significantly after being updated from a new resample.

The exact measure of convergence is given by the following algorithm: Given a series of realisations of ξ_n^*, $z^{(1)}, ..., z^{(t)} \in \mathbb{R}^d$; a window $w \in \mathbb{N}$ $(w < t)$; and a target relative error ϵ, we keep on producing subsamples until the following condition holds: $\forall j \in [1, w]$, $1/d \sum_{i=1}^{d} |(z_{(i)}^{t-i} - z_i^t|/|z_{(i)}^{(t)}| \leq \epsilon$.

The same condition can be used for the selection of s. Therefore, both parameters can be selected adaptively and in this case r is chosen independently for each subsample. The function `BLB.adapt` implements BLB with adaptive selection of both r and s. It takes data in the same format as `BLB.multi`, and likewise the parameter of interest is the coefficients of a penalised linear regression. Kleiner et al. (2014) suggest the values of $w = 3$ for the selection of s, $w = 20$ for the selection of r, and $\epsilon = 0.05$ for both adaptive procedures. The values of γ, λ and ϵ can also be set as arguments, as well as α, which controls the width of the confidence intervals. It returns the width of the marginal confidence intervals, along with the final value of s, and a vector of the values of r used in each subsample. Here we apply `BLB.adapt` to the same data as we used above with `BLB.multi`. Most arguments are set to the default values suggested above.

```r
set.seed(1)
system.time(
  BLB.adapt_out <- BLB.adapt(data, gamma=0.7)
)

##    user  system elapsed
##  16.458   3.922  20.380

BLB.adapt_out$s; BLB.adapt_out$r

## [1] 6
## [1] 46 69 60 51 57 76

BLB.adapt_out$mean_width

## [1] 0.09066808
```

We see that both s and r within each subsample are lower than the recommended defaults used in `BLB.multi`, and consequently `BLB.adapt` is much faster to run, whilst still returning a value of the mean width of the confidence intervals that is very close to those before. Moreover, the adaptive scheme avoids the problem of having to specify tuning parameters *a priori*, whilst still ensuring good statistical performance. We recommend always using `BLB.adapt` instead of `BLB.multi`.

The BLB is well-suited to modern parallel and computing distributed computing environments. It is possible to distribute subsamples to different computer nodes and then parallelize across resamples within each subsample. We have not pursued that line when developing and testing the `Bootstrap` package.

In our implementation of all three BLB functions, we at some point store all n values in each bootstrap replication. A more effecient implementation would avoid this. Since there are only $b(n)$ possible values in each resample (typically with $b(n) \ll n$), we should only need to know how many times each of these occur in the resample in order to compute $\hat{\theta}_n^*$. Our implementation is fast enough on the data we have simulated, but we expect that for massive datasets, we would encounter memory problems. The `BLB.multi` and `BLB.adapt` functions are also limited to regression problems where the parameter of interest are the coefficients. Future work includes a more generic implementation that can take a wider variety of data types and allow the user to specify a function for estimating any parameter of the underlying distribution.

References

Efron, Bradley (1979). “Bootstrap methods: another look at the jackknife”. In: *Annals of Statistics* 7, pp. 1–26.

Efron, Bradley and Gail Gong (1983). “A Leisurely Look at the Bootstrap, the Jackknife, and Cross-Validation”. In: *The American Statistician* 37.1, pp. 36–48.

Kleiner, Ariel et al. (2014). “A scalable bootstrap for massive data”. In: *Journal of the Royal Statistical Society (Series B)* 76.4, pp. 795–816.

Rubin, Donald B (1981). “The Bayesian Bootstrap”. In: *The Annals of Statistics* 9.1, pp. 130–134.