Paper 193-29

Bootstrap 101: Obtain Robust Confidence Intervals For Any Statistic

Dave P. Miller, Ovation Research Group, San Francisco, CA

ABSTRACT

For almost any statistic of interest, SAS/STAT PROCs generally contain options for obtaining a confidence interval. Some PROCs even provide multiple computational methods for estimating the standard errors and confidence intervals. In almost every case, however, the accuracy of the confidence intervals depends on parametric assumptions.

In such cases, bootstrap methods may be used to obtain a more robust non-parametric estimate of the confidence intervals. Bootstrap samples are very easy to generate using *SAS software*; however, it is a very computationally intensive method. In particular, the method is easy to apply in its most basic form even if you are not already familiar with bootstrap methods, as long as you are not already stretching the capabilities of your CPU and disk space.

The rationale for the bootstrap and the basics for interpreting the confidence intervals are explained through an example. The most efficient way to program and compute bootstrap confidence intervals depends in part on the size of the data set and the power of one's computer. Two different approaches are suggested depending on the limitations of ones data set and computing environment.

INTRODUCTION

For most SAS/STAT PROCs, confidence intervals are obtained based on a parametric estimate of the standard error (σ_θ) for the statistic of interest (θ). Generally, the 95% confidence interval is computed by adding or subtracting the standard error multiplied by a critical value (e.g. $\theta \pm 1.96\sigma_\theta$). This computation assumes that the confidence interval is symmetric around θ and that the estimate of σ_θ is correct.

There are many situations in which the parametric assumptions may be incorrect, and it is useful in such situations to compute bootstrap confidence intervals that do not rely on those assumptions. Distributional assumptions are commonly questioned in the presence of skewed data. Clustered data can also lead to incorrect assumptions about the error structure. Additionally, there are often statistics of interest that are a non-linear function of two or more potentially correlated statistics, and the confidence intervals for such statistics are not readily attainable using parametric methods.

When the parametric confidence intervals are of questionable merit, or difficult to obtain, it is possible to generate bootstrap samples and compute the statistic of interest for each bootstrap sample. The 2.5^{th} and 97.5^{th} percentiles of the bootstrap samples form a good approximation of the 95% confidence interval. This confidence interval may be compared to the parametric confidence interval as a sensitivity/robustness analysis, or in some cases it may be used as a substitute for the parametric confidence interval.

WHAT IS A BOOTSTRAP SAMPLE?

The general class of methods known as resampling procedures includes both the jackknife and the bootstrap. The bootstrap is a way of using the data collected for a single experiment to simulate what the results might be if the experiment was repeated over and over with a new sample of patients. These new simulated experiments are called bootstrap samples, and they are created by sampling with replacement from the original dataset.

In a particular bootstrap sample, a given subject from the original study may appear once, twice, many times, or not at all. This simulates what would happen if a new experiment were conducted. While the exact patient from the original study probably would not be in a new study, the number of very similar patients in the new study could be one, two, many, or none.

The Bootstrap was introduced and popularized by Efron (1979, 1982) and has been discussed in greater detail with many variations by other authors (Chernick, 1999). This paper focuses on the simplest form of the nonparametric bootstrap.

SAMPLE DATA SET

The sample data set has 200 records, one for each of 100 patients receiving treatment A (*tx*="A") and 100 patients receiving treatment B (*tx*="B"). In addition to the treatment assignment, each record also contains a binary variable, *event*, indicating whether or not the patient had an event and a continuous variable, cost, that represents the total cost over the treatment period.

On average, treatment A is more costly than treatment B and both distributions are heavily skewed. This is illustrated in the output from PROC MEANS below.

```
          Analysis Variable : cost

tx    Obs       Mean    Minimum    Maximum
ffffffffffffffffffffffffffffffffffffffffff
A     100      19108      10235     210274

B     100      11440       2046     236047
ffffffffffffffffffffffffffffffffffffffffff
```

On the other hand, events are much more common for treatment B, suggesting a likely cost-effectiveness tradeoff. That is, the fact that 80% of treatment A patients were event free compared to 60% of treatment B patients implies that the cost of preventing a single event could be estimated in dollars. This is illustrated in the PROC FREQ output below.

```
event     tx

Frequency,
Col Pct  ,A       ,B       ,  Total
fffffffffffffffffffffffffffff
       0 ,     80 ,     60 ,    140
         ,  80.00 ,  60.00 ,
fffffffffffffffffffffffffffff
       1 ,     20 ,     40 ,     60
         ,  20.00 ,  40.00 ,
fffffffffffffffffffffffffffff
Total         100      100      200
```

Specifically, based on the two sets of output above, the cost-effectiveness ratio can be computed as ($19,108-$11,440)/(80% – 60%) = $38,340 per event prevented. The standard error for the numerator is easy to compute; however, the fact that the costs are highly skewed may cause it to be somewhat inaccurate. More importantly, the standard error for the ratio is very messy to compute, even if strong assumptions are made about the correlation between costs and events.

GENERATING BOOTSTRAP SAMPLES

In the following example, temp01 is the data set containing the variables *cost*, *event*, and *tx*, but this code could be used to generate 1000 bootstrap samples for any data set.

```
* Create a sequential patid, numbered 1 to N - in this case N=200;

proc sort data=temp01 out=temp02;
  by patid;
run;

data temp03;
  set temp02;
  by patid;
  retain orig_seq_patid;
  if (_N_ eq 1) then
     orig_seq_patid=0;
  if first.patid then
     orig_seq_patid=orig_seq_patid+1;
  rename patid=orig_nonseq_patid;
run;

* Generate 1000 bootstrap samples of random patient IDs;

%let numsamp=1000;
%let numpat=200;

data temp04;
  do bootsamp=1 to &numsamp;
    do bootsamp_patid=1 to &numpat;
         random_seq_patid=ceil(&numpat*ranuni(1));
         output;
      end;
  end;
run;

* Link random patient IDs from bootstrap samples to the original data;

proc sql;
  create table temp05 as
     select distinct * from temp03 inner join temp04
       on temp03.orig_seq_patid=temp04.random_seq_patid
       order by bootsamp, bootsamp_patid;
quit;
```

Because this data set only has one record per patient, orig_seq_patid could have been assigned more easily for this particular example as orig_seq_patid=_N_, but the code that is used here could also be used without modification for a data set that had multiple records per patient.

COMPUTATION OF CIs USING BY-PROCESSING

The original data set had 200 records, so the new data set has 200,000 records. Because there are only a few variables involved in this analysis, a data set of this size is still manageable. Therefore, by-processing is an efficient way of programming the computation of the statistic of interest for each of the bootstrap samples.

```
%macro ce_ratio(indat=,outdat=,nsamps=);

  data _mac01;
    set &indat;
       if (1 le bootsamp le &nsamps);
  run;
```

```
  proc summary data=_mac01;
    where (tx eq "A");
       by bootsamp;
       var cost event;
       output out=_mac02
           mean=txa_avg_cost
                txa_event_rate;
  run;

  proc summary data=_mac01;
    where (tx eq "B");
       by bootsamp;
       var cost event;
       output out=_mac03
           mean=txb_avg_cost
                txb_event_rate;
  run;

  data &outdat;
    merge _mac02 _mac03;
       by bootsamp;
       costdiff=txa_avg_cost-txb_avg_cost;
       events_saved=(1-txa_event_rate)-
                    (1-txb_event_rate);
       if (events_saved gt 0) then
        ce_ratio=costdiff/events_saved;
       * If no events saved, set ce ratio
         to arbitrarily large number;
       else ce_ratio=100000000;
  run;

  proc datasets lib=work;
    delete _mac01 _mac02 _mac03;
  run;

%mend ce_ratio;

%ce_ratio(indat=temp05, outdat=temp06, nsamps=1000);
```

The output data set, temp06, has 1000 observations, one for each of the 1000 bootstrap samples. Note that the code for computing the cost-effectiveness ratio for the original sample would look exactly the same as this code, but without the *by bootsamp* lines in each PROC and DATA step.

Having computed the statistic of interest for each of the 1000 bootstrap samples, the final step is to compute the confidence intervals. Both the 95% CI and the 70% CI are computed using the code below.

```
proc univariate loccount data=temp06;
  var ce_ratio costdiff events_saved;
  output out=temp07 n=n_samples
         pctlpre=ce_ci_ pctlpts=2.5,97.5,15,85;
run;

proc print noobs label data=temp07;
  title1 "Bootstrap 95% and 70% CIs";
  var n_samples ce_ci_2_5 ce_ci_97_5;
  var n_samples ce_ci_15 ce_ci_85;
  format ce_ci_2_5 ce_ci_97_5 ce_ci_15 ce_ci_85 dollar12.;
run;
```

The output from the PROC PRINT shows the two confidence intervals, and also illustrates the high degree of uncertainty at the upper end of the range.

Bootstrap 95% and 70% CIs

number of nonmissing values, ce_ratio	the 2.5000 percentile, ce_ratio	the 97.5000 percentile, ce_ratio
1000	$5,420	$113,281

number of nonmissing values, ce_ratio	the 15.0000 percentile, ce_ratio	the 85.0000 percentile, ce_ratio
1000	$20,666	$63,745

In this example, the cost of preventing a single event may be as little as $5,000 or it may be more than $100,000.

COMPUTATION OF CIs USING APPEND

In the case study presented here, the statistic that was being computed did not require us to retain a large number of variables and the size of the data set was very manageable. This is not always true. Sometimes, it is therefore more efficient to compute the statistic for each bootstrap sample and store the estimate before computing the statistic for the next sample.

```
%macro compute_and_append(caa_indat=, caa_outdat=, firstsamp=1, lastsamp=);

  data _caa01;
    set &caa_indat;
       if (bootsamp eq &firstsamp);
  run;

  %ce_ratio(indat=_caa01, outdat=_caa02, nsamps=1);

  proc datasets lib=work;
     delete _caa01;
  run;

  x "mkdir c:\boot_tracking";

  data _null_;
       file "c:\boot_tracking\track.log" mod;
       put "Done processing bootstrap sample 1";
  run;

  %do currsamp=&firstsamp+1 %to &lastsamp;

    data _caa01;
      set &caa_indat;
         if (bootsamp eq &currsamp);
    run;

    %ce_ratio(indat=_caa01, outdat=_caa03, nsamps=&currsamp);

       proc append base=_caa02 data=_caa03;
       run;

       proc datasets lib=work;
          delete _caa01 _caa03;
       run;
```

```
        data _null_;
           file "c:\boot_tracking\track.txt" mod;
           put "Done processing bootstrap sample " @;
         put "&currsamp";
         run;

    %end;

    data &caa_outdat;
      set _caa02;
    run;

    proc datasets lib=work;
      delete _caa02;
    run;

%mend compute_and_append;

%compute_and_append(caa_indat=temp05, caa_outdat=temp08, firstsamp=1,
lastsamp=1000);

proc univariate loccount data=temp08;
  var ce_ratio costdiff events_saved;
  output out=temp09 n=n_samples
         pctlpre=ce_ci_ pctlpts=2.5,97.5,15,85;
run;

proc print noobs label data=temp09;
  title1 "Alt Method: Bootstrap 95% and 70% CIs";
  var n_samples ce_ci_2_5 ce_ci_97_5;
  var n_samples ce_ci_15 ce_ci_85;
  format ce_ci_2_5 ce_ci_97_5 ce_ci_15 ce_ci_85 dollar12.;
run;
```

Because the bootstrap samples were selected in a previous step, the confidence intervals using the compute and append approach are identical to those using the previously described by-processing approach. Unfortunately, the compute and append approach can create log files that are too large for the log window in interactive SAS, so it is often better to run these programs in batch mode.

Because the programs take such a long time to run, it is natural for the curious (i.e. obsessive) analyst to want to monitor the progress of the program. This program creates a file called c:\boot_tracking\track.txt and appends that file with a single line each time a bootstrap sample is done being processed. The file can be monitored simply by using the type command in an MS-DOS window.

DISCUSSION

The bootstrap is a powerful tool for testing or avoiding parametric assumptions when computing confidence intervals. Although it is a computationally intensive method, it is the CPU time more than the programmer's time that leads to that characterization.

As noted previously, the first step of the process, described in the section titled bootstrap samples can be applied to almost any problem and any data set. For instance, if a lab value or quality of life survey was collected for every patient at each of three time intervals, the FIRST.PATID processing and the PROC SQL code for creating the complete set of bootstrap samples cause all patient data to be approapriately selected as a clustered unit.

The second step is to compute the statistic. It is recommended that this be programmed first just using the original data. For instance, the ce_ratio could be computed as follows:

```
data temp11;
  set temp01;
  origsamp=1;
run;
```

```
proc summary data=temp11;
  where (tx eq "A");
  by origsamp;
  var cost event;
  output out=temp12
         mean=txa_avg_cost
              txa_event_rate;
  run;

proc summary data=temp11;
  where (tx eq "B");
  by origsamp;
  var cost event;
  output out=temp13
         mean=txb_avg_cost
              txb_event_rate;
run;

data temp14;
  merge temp12 temp13;
  by origsamp;
  costdiff=txa_avg_cost-txb_avg_cost;
  events_saved=(1-txa_event_rate)-
               (1-txb_event_rate);
  if (events_saved gt 0) then
     ce_ratio=costdiff/events_saved;
  * If no events saved, set ce ratio
    to arbitrarily large number;
  else ce_ratio=100000000;
run;

proc print data=temp14;
  var origsamp ce_ratio;
run;
```

To convert this code to a macro, one need only rename the input and output data sets and replace *origsamp* with *bootsamp*. Once this has been done, it is very useful to test the macro using only 5 or 10 bootstrap samples. If one bootstrap sample yields a much higher or much lower value than the others, it can be helpful to work backwards and see if that sample was one in which an outlier was either absent or sampled multiple times. Programming in this type of stepwise manner will help avoid mistakes, make it easier to debug the program when mistakes do occur, and also help the programmer develop intuition about the bootstrap method.

CONCLUSION

If you have not used the bootstrap method previously, it is best to begin using by comparing the bootstrap results to those from a parametric model. If you have a large sample and the assumptions are at least reasonable, albeit imperfect, the bootstrap confidence interval will be similar to the parametric confidence interval. If this is not the case, it is quite possible that there is a problem with your code. While the bootstrap is relatively easy to program, it is more cumbersome to debug, due to the sheer volume of data created by simulating hundreds or thousands of experiments. Once one understands the power of the bootstrap methods, there is a temptation to use it for everything, but it should be considered a complimentary tool, not a replacement for parametric methods.

REFERENCES

Efron B (1979). Bootstrap methods: another look at the jackknife. Ann. Statist. **7**, 1-26.
Efron B (1982). *The Jackknife, the Bootstrap, and Other Resampling Plans*. SIAM, Philadelphia.
Chernick MR (1999). *Bootstrap Methods: A Practioner's Guide*. John Wiley & Sons.

CONTACT INFORMATION

dmiller@ovation.org