Bonsai

Statistical Benchmarking in J

Table of Contents

This is a `J` program which does some statistical analysis on computer benchmark results. This is also a collection of notes so that I may recall what I learned while writing this script.

I wanted a way to know which changes I was making to `J` programs were actually effective. The built in time function `x 6!:2 y`, which averages the time taken to run `y x` times, isn't great for this purpose as averaging destroys information and it takes a hit from initial interpretive overhead. Light programs `y` thus require a large `x`, before the results converge.

`bonsai` uses is statistical bootstrapping, which enables us to estimate statistical parameters on nonparametric samples. I got the sense of what to look for from haskell's criterion package, and learned the material for how to implement it through Efron and Hastie's textbook Computer Age Statistical Inference. See chapters 10 and 11 from there to get the original material.

Background

We take a sample $x = (x_i)$ of benchmark results, which we (slightly dubiously) assume iid from some unknown distribution. From that we compute various statistics on that sample from corresponding algorithms $\hat{\theta} = s(x)$ in order to understand the results.

Some of these benchmarks will be expensive to run and in general it won't be possible to gather a large sample. To that end, the process of statistical bootsrapping allows us to extrapolate better estimates on the desired statistics through resampling uniformly from x. Moreover, this allows us to

calculate standard errors and to attach confidence intervals on these computed statistics. And for our purposes, perhpas the most crucial aspect is that nothing need be assumed or known about the underlying distrubiton and that the computations are automatic.

Bootstrapping

A single bootstrap resample of the original sample x is $x^* = (x_i^*)$ where the x_i* are drawn uniformly from x and $n = |x| = |x^*|$. This resampling is carried out B times, comprising the bootstrapped resample on which we compute statistics $\hat{\theta}^{*b}$. And from that, statistics such as

$$\hat{\text{se}} = \sqrt{\frac{\sum_b \left(\hat{\theta}^{*b} - \hat{\theta}^{*\cdot}\right)^2}{B-1}}$$

can be computed. Basically, the process is getting a sample x from some unknown distribution F, and computing a statistic $\hat{\theta}$ on it. Then, better estimates of $\hat{\theta}$ come from resampling x^* from the emperical distribution $\hat{F}$ which assigns probability $\frac{1}{n}$ to each x_i from x then calculating $\hat{\theta}^*$. The key to this process is that $\hat{F} \to F$ as $n \to \infty$, and that for any n, $\hat{F}$ maximizes the probability of having observed x from F, whatever F may actually be. In other words, $\hat{F}$ is the nonparametric maximum likelihood estimator for F.

The final step in the above process is akin to running another algorithm $\text{Sd}(\hat{F}) = \hat{\text{se}}$ on the bootstrap resample and is called the ideal bootstrap estimate of the standard error. We can, however, do better inference and construct confidence intervals on the $\hat{\theta}^*$.

Before describing the bootstrap confidence calculations, a diversion on configuration.

Initial Sample and Configuration

The basic configuration is the amount of time alloted for the initial benchmarks, the minimum number of runs, and the maximum number runs. Additional configuration includes the target coverage for confidence intervals and the number of bootstrap trials.

```
time  =: 1      NB. time alloted (upper bound on)
lo    =: 5      NB. minimum sample
hi    =: 2000   NB. maximum sample
alpha =: 0.05   NB. coverage
B     =: 2500   NB. bootstrap resample
```

We gather an initial sample from `dobench` by first running the program once and run it a number of times based on the configuration. It's also possible to specify exactly how many times to run a benchmark by using it dyadically.

```
dobench=:  1 : 0
NB. u dobench y: run sentence y a number of times based on the
```

```
NB. configuration. u is the locale where the sentence was called from,
NB. which is captured in the bonsai verb.
 cocurrent u
 t0 =. (hi_bonsai_ <. lo_bonsai_ >. >. time_bonsai_ % 1e_6 >. 6!:2 y)
 xs =. 6!:2"1 t0 # ,: y
 cocurrent 'bonsai' NB. apparently u would otherwise stick during
                    NB. execution of other verbs (eg summarize)
 xs
)

dobootstrap=: 1 : 0
NB. u doboostrap: redraw uniformly from sample y u times.
 y {~ ? u # ,: $~ #y
)
```

Percentiles

Discrete percentiles and quantiles are not in J's stats addon and can be computed as follows:

```
qtile =: 4 : 0
 ws=. (%+/)"1 -. | xs -"0 1 is=. (<.,>.)"0 xs=. x * <:#y
 ws (+/"1 @: *) is { /:~ y
)

cdf =: (+/ @: (<:/~) % #@]) :. qtile

IQR =: -/ @: (0.75 0.25&qtile)
```

The local variables `is` and `ws` in `qtile` are used to interpolate between values at neighboring indices so that for example `0.5 (cdf^:_1) 0 3` and `median 0 3` agree and are both `1.5`. The obverse counts how many elements of `y` are less than or equal to `x`. `IQR` is the interquartile range.

KDE

For visualization purposes, here is some basic functionality for calculating kernel density estimates given a benchmark.

```
bandwidth =: 0.9 * (stddevp * 5 %: 4r3 % #) NB. <. (0.746269 * IQR)

kde =: 2 : 0
NB. n sample, u kernel, y point
 (h%#n) * +/ u (y - n) * h =. % bandwidth n
)
NB. standard normal pdf & epanechnikov kernels
phi =: (%:%2p1) * [: ^ _0.5 * *:
epanechnikov =: 3r4 * 0 >. 1 - *:
```

Bootstrapping Confidence

Corresponds to Chapter 11 of casi textbook. Throughout, goal is to estimate the unseen statistic θ from the bootstrap resample $\hat{\theta}^*$

Standard Interval

The simplest but least accurate way of stamping a condience interval on the resampled statistics $\hat{\theta}^*$ is by taking the bootstrapped standard error and asking for coverage based on the normal distribution cdf.

```
bssi=: 1 : 0
NB. x u bspi y: verb u is statistic, y is sample, x is resample.
 (mean s) -`[`+`:0 (stddev s=. u"1 x) * qnorm -. -: alpha
)
```

In other words for 95% coverage the estimate for θ is inside interval $\hat{\theta} \pm 1.96 \cdot \hat{\text{se}}$. 1.96 comes from cdf of standard normal distribution $\Phi^{-1}(0.975)$. The 0.975 comes from $1 - \frac{\alpha}{2}$ and our α is configured through the variable `alpha`.

Percentile Interval

The next best way to go is to use percentiles on the emperical resamples to find our confidence.

```
bspi=: 1 : 0
NB. x u bspi y: verb u is statistic, y is sample, x is resample.
 ((-:i.3) + (i:_1) * -:alpha) cdf^:_1 u"1 x
)
```

In other words, we estimate θ from the bootstrap cdf $\hat{F}$, and get the interval $\hat{F}^{-1}[\frac{\alpha}{2}, 1 - \frac{\alpha}{2}]$. In J the base interval is cutely calculated by hooking `(,-.) -: alpha`.

Bias-corrected Percentile Interval

The resamples may skew more heavily to one side or the other of $\hat{\theta}$. To correct for this, we look at the percentile of the it in the resample then derive the bounds on the confidence interval by mapping through the standard normal cdf Φ getting the desired coverage and then calculating percentiles.

```
bsbc=: 1 : 0
NB. x u bsbc y: verb u is statistic, y is sample, x is resample.
 that =. u samp =. y
 z0=. qnorm that cdf resamp =. u"1 x
 I=. pnorm (+: z0) + qnorm (,-.) -: alpha
 ({.,that,{:) I (cdf^:_1) resamp
)
```

The above corresponds to

$$p_0 = \frac{\#\{\hat{\theta}^{*b} \leq \hat{\theta}\}}{B}$$

$$z_0 = \Phi^{-1}(p_0)$$

$$\hat{\theta}_{\mathrm{BC}}[\alpha] = \hat{F}^{-1}[\Phi(2 \cdot z_0 + z^{(\alpha)})]$$

When the bootstrap resamples are median unbiased (ie $p_0 = 0.5$) then $z_0 = 0$ and this agrees with the simple percentile interval.

Bias-corrected and Accelerated Percentile Interval

The previous method assumes the existence of a monotone transform $\hat{\phi} = m(\hat{\theta})$ such that $\hat{\phi} \sim N(\hat{\phi} - z_0\sigma, \sigma^2)$. The standard error is assumed constant. Relaxing the assumption to let it vary with ϕ is the key to the accelerated method. We assume the error is described by some acceleration a in

$$\hat{\phi} \sim N(\phi - z_0\sigma_\phi, \sigma_\phi^2) \text{ , with } \sigma_\phi = 1 + a\phi$$

```
bsbca=: 1 : 0
NB. x u bsbca y: verb u is statistic, y is sample, and x is resample.
 thati=. (1 u \. y) - that =. u y
 ahat=. 1r6 * (+/thati^3) % (+/*:thati)^3r2
 z0qt=. that cdf resamp=. u"1 x
 ab =. (,-.) -: alpha
 if. 1 ~: ab I. z0qt do. x u bspi y
 else. z0=. qnorm z0qt
       zabh=. z0 + (% 1 - ahat&*) z0 + qnorm ab
       ({.,that,{:) (pnorm zabh) cdf^:_1 resamp
 end.
)
```

The above corresponds to calculating

$$\hat{\theta}_{\mathrm{BCa}}[\alpha] = \hat{F}^{-1}\left[\Phi\left(z_0 + \frac{z_0 + z^{(\alpha)}}{1 - a(z_0 + z^{(\alpha)})}\right)\right]$$

where the a term is found by jack-knifing the statistic θ on the original sample in unbiasing by its skewness.

Description

Regression

J programs don't tend to have much overhead, but this is a nice idea from criterion. One way to estimate the performance of a program is to do a linear regression on the sample. Presumably the overhead will be captured in the constant term, giving a clearer picture of typical execution times. Here, we sum of the execution times to get `n` snapshots of performance.

```
regress_bench=: +/\ %. 1 ,. i.@#
rsquare_bench=: 3 : 0
 b=. (y=.+/\y) %. v=. 1,.i.#y
 (sst-+/*:y-v +/ . * b)% sst=. +/*:y-(+/y) % n=. #y
)
```

Bootstrap-t

Find confidence for $\theta = \mu_x - \mu_y$ given two samples of size n_x and n_y. Estimate $\hat{\theta} = \bar{x} - \bar{y}$. Depends on nuissance parameter σ^2. Traditional student-t instead bases $\hat{\theta}$ on pivotal quantity $t = \frac{\hat{\theta}-\theta}{\hat{se}}$. $\hat{se}$ is unbiased estimater for nuissance parameter

$$\hat{se}^2 = \left(\frac{1}{n_x} + \frac{1}{n_y}\right) \cdot \frac{\sum(x-\bar{x})^2 - \sum(y-\bar{y})^2}{n_x + n_y - 2}$$

Bootstrap-t instead estimates distribution of t through bootstrapping. Nonparametric resamples are drawn from x and y, $\hat{\theta}$ plays the role of our assumption $\mu_x - \mu_y$, and we examine $t^* = \frac{\hat{\theta}^* - \hat{\theta}}{\hat{se}^*}$. The quantiles from the replications t^{*b} provide the confidence intervals

$$\hat{\theta}^*[\alpha] = \hat{\theta} - \hat{se} \cdot t^{*(1-\alpha)}$$

In J:

```
se2_t=: +&%&# * +&ssdev % +&#-2:
se_t=: %:@:se2_t

bs_t=: 4 : 0
NB. x bs_t y: use bootstrap-t to compare distributions of benchmark
NB. results form sentences x and y.
 that=. x -&mean y
 sehat=. x se_t y
 sx =. B dobootstrap x
 sy =. B dobootstrap y
 xbar =. sx mean bs_est x
 ybar =. sy mean bs_est y
 dsamp=. sx ((that -~ -&mean) % se_t)"1 sy
 ths =. ({.,that,{:) that - sehat * ((,~-.) -: alpha) cdf^:_1 dsamp
 xvy =. xbar (-~%[) ybar
 ths , xbar , ybar ,: xvy
)

bs_compare=: bs_t & dobench
```

The idea is we can get some confidence on the parameter $\hat{\theta} = \bar{x} - \bar{y}$ of the two samples by taking μ_x, μ_y from the original sample, then bootstrapping the pivotal quantity $t*$.

Analysis

Verb `bonsai` defaults to using `bsbca` and estimate some descriptive statistics in `summarize`. `bonsai` is ambivalent and when used as a dyad benchmarks two sentences, comparing their mean execution times via bootstrap-t. As a monad, it outputs some descriptive statistics.

```
NB. use bs bias corrected accelerated by default
bs_est =: bsbca

mu =: u: 16b3bc
delta =: u: 16b3c3

summarize =: 3 : 0
NB. Report some descriptive statistics about a list y of benchmark results.
 resamp=. B dobootstrap samp=. y
 xbarc=. resamp mean bs_est samp
 sdevc=. resamp stddev bs_est samp
NB. regac=. resamp ({:@regress_bench) bs_est samp
NB. rsqrc=. resamp rsquare_bench bs_est samp
NB.  skwnc=. resamp skewness bs_est samp
NB.  kurtc=. resamp kurtosis bs_est samp
 ests=. <"0 xbarc ,: sdevc NB. , regac ,: rsqrc
 ests=. (;: 'lower estimate upper') , ests

 rows=. ('N = ',":#samp);mu;delta NB. ;'ols';('R',(u:16bb2),' (ols)')
 rows ,. ests
)

bonsai3=: 1 : 'summarize u dobench_bonsai_ y'
bonsai4 =: 1 : 0
 table =. ;: 'comparison lower estimate upper'
 x =. u dobench_bonsai_ x
 y =. u dobench_bonsai_ y
 x_y =. x bs_t y
 table =. table , (('- & ',mu);(mu,'(x)');(mu,'(y)');'x (-~%[) y') ,. <"0 x_y
)

bonsai_z_ =: 3 : 0
NB. bonsai y: estimate performance of sentence y
  loc =. coname''
  loc bonsai3_bonsai_ y
:
NB. x bonsai y: compare mean execution time of two sentences. a
NB. positive estimate means sentence x takes longer to execute than
NB. sentence y.
  loc =. coname ''
  x loc bonsai4_bonsai_ y
)
```

Printing Times

```
bsppns =: 'ns' ,~ [: ": [: <. 0.5 + 1e9&*
bsppus =: ('s',~u:16b3bc) ,~ [: ": [: <. 0.5 + 1e6&*
bsppms =: 'ms' ,~ [: ": [: <. 0.5 + 1e3&*
bspps =: 's' ,~ [: ": (100 %~ [: <. 0.5 + 100&*)
bsppa =: bsppns`bsppus`bsppms`bspps@.(_6 _3 0 I. 10&^.)
bsnump =: 1 4 8 e.~ 3!:0
bsucp =: 131072 262144 e.~ 3!:0
bspp =: bsppa ^: bsnump

bonsaipp =: 3 : 0
 NB. with monadic bonsai usage, pretty print the results
 res =. bonsai y
 (u:@":) ^: (-.@bsucp) &.> ({: res) ,~ bspp &.> }: res
)
```

Plotting

```
bonsaiplot_z_ =: 3 : 0
NB. plot kde of benchmark of sentence y
 pd 'reset; visible 0;title ',y
 pd 'xcaption time;ycaption kde & sample over time'
 'a b' =. (<./,>./) samp =. (coname'') dobench_bonsai_ y
 den =. (epanechnikov kde samp)"0 pts =. (-:a+b) + ((%~i:)1000) * 0.6*b-a
 pd 'type dot;pensize 0.3;color 80 100 200'
 pd samp;(>./den)*(%~i.)#samp
 pd 'type line; pensize 2;color 0 120 240'
 pd pts;den
 pd 'key sample density; keycolor 80 100 200,0 120 240;keypos top right'
 pd'show'
)
```

Final Program

```
coclass 'bonsai'
load 'stats/base stats/distribs'

time  =: 1      NB. time alloted (upper bound on)
lo    =: 5      NB. minimum sample
hi    =: 2000   NB. maximum sample
alpha =: 0.05   NB. coverage
B     =: 2500   NB. bootstrap resample

dobench=:  1 : 0
NB. u dobench y: run sentence y a number of times based on the
NB. configuration. u is the locale where the sentence was called from,
NB. which is captured in the bonsai verb.
 cocurrent u
 t0 =. (hi_bonsai_ <. lo_bonsai_ >. >. time_bonsai_ % 1e_6 >. 6!:2 y)
```

```
 xs =. 6!:2"1 t0 # ,: y
 cocurrent 'bonsai' NB. apparently u would otherwise stick during
                    NB. execution of other verbs (eg summarize)
 xs
)

dobootstrap=: 1 : 0
NB. u doboostrap: redraw uniformly from sample y u times.
 y {~ ? u # ,: $~ #y
)

qtile =: 4 : 0
 ws=. (%+/)"1 -. | xs -"0 1 is=. (<.,>.)"0 xs=. x * <:#y
 ws (+/"1 @: *) is { /:~ y
)

cdf =: (+/ @: (<:/~) % #@]) :. qtile

IQR =: -/ @: (0.75 0.25&qtile)

bandwidth =: 0.9 * (stddevp * 5 %: 4r3 % #) NB. <. (0.746269 * IQR)

kde =: 2 : 0
NB. n sample, u kernel, y point
 (h%#n) * +/ u (y - n) * h =. % bandwidth n
)
NB. standard normal pdf & epanechnikov kernels
phi =: (%:%2p1) * [: ^ _0.5 * *:
epanechnikov =: 3r4 * 0 >. 1 - *:

bssi=: 1 : 0
NB. x u bspi y: verb u is statistic, y is sample, x is resample.
 (mean s) -`[`+`:0 (stddev s=. u"1 x) * qnorm -. -: alpha
)

bspi=: 1 : 0
NB. x u bspi y: verb u is statistic, y is sample, x is resample.
 ((-:i.3) + (i:_1) * -:alpha) cdf^:_1 u"1 x
)

bsbc=: 1 : 0
NB. x u bsbc y: verb u is statistic, y is sample, x is resample.
 that =. u samp =. y
 z0=. qnorm that cdf resamp =. u"1 x
 I=. pnorm (+: z0) + qnorm (,-.) -: alpha
 ({.,that,{:) I (cdf^:_1) resamp
)

bsbca=: 1 : 0
NB. x u bsbca y: verb u is statistic, y is sample, and x is resample.
 thati=. (1 u \. y) - that =. u y
 ahat=. 1r6 * (+/thati^3) % (+/*:thati)^3r2
 z0qt=. that cdf resamp=. u"1 x
 ab =. (,-.) -: alpha
 if. 1 ~: ab I. z0qt do. x u bspi y
 else. z0=. qnorm z0qt
```

```
        zabh=. z0 + (% 1 - ahat&*) z0 + qnorm ab
        ({.,that,{:) (pnorm zabh) cdf^:_1 resamp
 end.
)

regress_bench=: +/\ %. 1 ,. i.@#
rsquare_bench=: 3 : 0
 b=. (y=.+/\y) %. v=. 1,.i.#y
 (sst-+/*:y-v +/ . * b)% sst=. +/*:y-(+/y) % n=. #y
)

se2_t=: +&%&# * +&ssdev % +&#-2:
se_t=: %:@:se2_t

bs_t=: 4 : 0
NB. x bs_t y: use bootstrap-t to compare distributions of benchmark
NB. results form sentences x and y.
 that=. x -&mean y
 sehat=. x se_t y
 sx =. B dobootstrap x
 sy =. B dobootstrap y
 xbar =. sx mean bs_est x
 ybar =. sy mean bs_est y
 dsamp=. sx ((that -~ -&mean) % se_t)"1 sy
 ths =. ({.,that,{:) that - sehat * ((,~-.) -: alpha) cdf^:_1 dsamp
 xvy =. xbar (-~%[) ybar
 ths , xbar , ybar ,: xvy
)

bs_compare=: bs_t & dobench

bsppns =: 'ns' ,~ [: ": [: <. 0.5 + 1e9&*
bsppus =: ('s',~u:16b3bc) ,~ [: ": [: <. 0.5 + 1e6&*
bsppms =: 'ms' ,~ [: ": [: <. 0.5 + 1e3&*
bspps =: 's' ,~ [: ": (100 %~ [: <. 0.5 + 100&*)
bsppa =: bsppns`bsppus`bsppms`bspps@.(_6 _3 0 I. 10&^.)
bsnump =: 1 4 8 e.~ 3!:0
bsucp =: 131072 262144 e.~ 3!:0
bspp =: bsppa ^: bsnump

bonsaipp =: 3 : 0
 NB. with monadic bonsai usage, pretty print the results
 res =. bonsai y
 (u:@":) ^: (-.@bsucp) &.> ({: res) ,~ bspp &.> }: res
)

bonsaiplot_z_ =: 3 : 0
NB. plot kde of benchmark of sentence y
 pd 'reset; visible 0;title ',y
 pd 'xcaption time;ycaption kde & sample over time'
 'a b' =. (<./,>./) samp =. (coname'') dobench_bonsai_ y
 den =. (epanechnikov kde samp)"0 pts =. (-:a+b) + ((%~i:)1000) * 0.6*b-a
 pd 'type dot;pensize 0.3;color 80 100 200'
 pd samp;(>./den)*(%~i.)#samp
 pd 'type line; pensize 2;color 0 120 240'
 pd pts;den
```

```
 pd 'key sample density; keycolor 80 100 200,0 120 240;keypos top right'
 pd'show'
)

NB. use bs bias corrected accelerated by default
bs_est =: bsbca

mu =: u: 16b3bc
delta =: u: 16b3c3

summarize =: 3 : 0
NB. Report some descriptive statistics about a list y of benchmark results.
 resamp=. B dobootstrap samp=. y
 xbarc=. resamp mean bs_est samp
 sdevc=. resamp stddev bs_est samp
NB. regac=. resamp ({:@regress_bench) bs_est samp
NB. rsqrc=. resamp rsquare_bench bs_est samp
NB.  skwnc=. resamp skewness bs_est samp
NB.  kurtc=. resamp kurtosis bs_est samp
 ests=. <"0 xbarc ,: sdevc NB. , regac ,: rsqrc
 ests=. (;: 'lower estimate upper') , ests

 rows=. ('N = ',":#samp);mu;delta NB. ;'ols';('R',(u:16bb2),' (ols)')
 rows ,. ests
)

bonsai3=: 1 : 'summarize u dobench_bonsai_ y'
bonsai4 =: 1 : 0
 table =. ;: 'comparison lower estimate upper'
 x =. u dobench_bonsai_ x
 y =. u dobench_bonsai_ y
 x_y =. x bs_t y
 table =. table , (('- & ',mu);(mu,'(x)');(mu,'(y)');'x (-~%[) y') ,. <"0 x_y
)

bonsai_z_ =: 3 : 0
NB. bonsai y: estimate performance of sentence y
  loc =. coname''
  loc bonsai3_bonsai_ y
:
NB. x bonsai y: compare mean execution time of two sentences. a
NB. positive estimate means sentence x takes longer to execute than
NB. sentence y.
  loc =. coname ''
  x loc bonsai4_bonsai_ y
)
```

Created: 2021-01-05 Tue 14:29

[Validate]