Notes on resampling and permutations, F. Chiaromonte Sp 06

Boostrap (SE and CI):

Consider a statistic such as the mean or the median of y (univariate, i.e. concerning one variable), or the Pearson correlation between y and x (bivariate, i.e. concerning two variables), or the difference between the mean of y in two groups (bivariate, too; concerns y and the categorical variable defining the groups), or the jth eigenvalue of the var/cov matrix of $\underline{x}=(x_1 \ldots x_p)$ (multivariate)

In order to interpret the observed value of a statistic, we need to go beyond thinking of it as a summary descriptor of the data. Suppose the data is a random sample of size *n* from a population. Let *F* indicate the distribution of the variable(s) in the population, and F_n indicate the distribution on the sample, or empirical distribution. This places probability 1/*n* on each of the observed values, and has the nice property of approximating *F* more and more closely as n increases. If we are interested in a certain feature of the population $t(F)$, we can estimate it on the data as $t(F_n)$ – which represents the "plug-in" sample equivalent of $t(F)$ (for instance, we could estimate the population median with the sample median, or the population correlation with the sample correlation, or the population jth var/cov eigenvalue with the sample jth var/cov eigenvalue):

The estimate $t(F_n)$ can be thought of as a random quantity (over all possible *n*-samples) of which we observe one possible determination (on the sample we drew). As such, it will have a sampling distribution $P_F^{(n)}$ that hopefully relates nicely to the population quantity $t(F)$; for instance we look for:

$$E(t(F_n)) = \int t P_F^{(n)}(dt) \quad = t(F)$$

$$E(t(F_n)) - t(F) = Bias(t(F_n)) \xrightarrow[n\to\infty]{} 0$$

$$MSE(t(F_n)) = \int (t - t(F))^2 P_F^{(n)}(dt) \xrightarrow[n\to\infty]{} 0$$

$$[\ MSE(t(F_n)) \cong \mathrm{var}(t(F_n)) = \int (t - E(t(F_n)))^2 P_F^{(n)}(dt) \ \text{ if } \ Bias(t(F_n)) \cong 0\ ]$$

$$t(F_n) \xrightarrow[n\to\infty]{} t(F)$$

- Unbiased (sample mean for popul mean, but not "n-1" sample var for popul var).
- Bias vanishing for large samples; mse and var approx the same ("n-1" sample var for popul var).
- MSE vanishing for large samples gives consistency.

We define the **STANDARD ERROR** of the statistic $t(F_n)$ as an estimate of the square root of its MSE in estimating $t(F)$ with a finite sample size. If the statistic carries no bias (a vanishing bias) this is equivalently (approximately) an estimate of the standard deviation of the statistic (square root of it variance).

Being able to compute the standard error of a statistic is crucial, as it benchmarks its observed value against the variability the statistic presents vis a vis the population quantity it is trying to estimate.

The (non-parametric) **BOOSTRAP** is a computational tool to "simulate" the sampling distribution $P_F^{(n)}$, and thus to produce an approximate standard error, for any statistic. Here is the basic logic:

For b=1…B

1. Create a size n bootstrap sample drawing with replacement from F_n (i.e. resample the available data with replacement). This gives (y_1*(b)… y_n*(b)) or F_n*(b)

2. Compute the statistic on it $t(F_n$*(b)) (e.g. med*(b) = median of ((y_1*(b)… y_n*(b)))

Approximate the MSE ideal bootstrap expression, and thus the produce the bootstrap se:

$$\tilde{MSE}_{boot}(t(F_n)) = \frac{1}{B}\sum_{b=1\ldots B}(t(F_n^*(b)) - \bar{t})^2 \quad \text{with} \quad \bar{t} = \frac{1}{B}\sum_{b=1\ldots B} t(F_n^*(b))$$

$$se_{boot}(t(F_n)) = \sqrt{\tilde{MSE}_{boot}(t(F_n))}$$

Depending on the complexity of the statistic, B in the range of 50-200 already gives a reliable approximation of the standard error.

The following is an example of (non-parametric) bootstrap for the median in R. The variable we consider is y = log-length ratio in as example data set (chicken_toy.txt) with sample size n=100. We use B=200.

```
> library(boot)                              #loads the boot package

> chicken_toy <- read.table("chicken_toy.txt",header=TRUE)

> #The boot() function requires in input a function expressing
> #the statistic in a special way, i.e. with two required
> #arguments, so even if median() exists already as a function
> #in R, we create a user-defined function:

> med_fun <- function(data,indices){
+ median(data[indices])                     #last line: output
+ }

> #Note: user-defined functions are stored in the workspace
> ls()
[1] "med_fun"
> #and can be viewed
> med_fun                                    #shows on the console
> #or edited
> edit(med_fun)                              #pop-up text editor

> boot_loglenrat <-
boot(data=chicken_toy[,"y"],statistic=med_fun,R=200)

> #Optional argument sim="ordinary","parametric"… for non-
> #parametric, parametric, other more sophisticated bootstrap
> #sampling schemes. "ordinary" is default. Many more optional
> #arguments (many things can be done with bootstrap!)
> boot_loglenrat
```

```
ORDINARY NONPARAMETRIC BOOTSTRAP

Call:
boot(data = chicken_toy[, "y"], statistic = med_fun, R = 200)

Bootstrap Statistics :
    original      bias    std. error
t1* 0.358885 0.000228975  0.01842007

> #value of the statistic on the data, and bootstrap estimated
> #standard error, as well as bias. Note: boot_loglenrat
> #actually contains the 200 bootstrap values of the statistics
> boot_loglenrat["t"]

$t
           [,1]
  [1,] 0.346310
  [2,] 0.377465
       .
       .
       .
  [199,] 0.357930
  [200,] 0.380210
```

The bootstrap can also be used to create a **CONFIDENCE INTERVAL** for the population feature $t(F)$. There are a wide variety of bootstrap-based approaches, with various levels of sophistication. Here we introduce only a very simple one, the (non-parametric) bootstrap-based percentile interval. Here is the basic logic:

To create a confidence interval with coverage $1-2\alpha$, consider the bootstrap distribution of the statistics (our simulated proxi for $P_F^{(n)}$) and create an interval "cutting off" α-mass from the left and the right tail; that is

For b=1…B

1. Create a size n bootstrap sample drawing with replacement from F_n (i.e. resample the available data with replacement). This gives $(y_1^*(b)\ldots y_n^*(b))$ or $F_n^*(b)$.

2. Compute the statistic on it $t(F_n^*(b))$ (e.g. med*(b) = median of $((y_1^*(b)\ldots y_n^*(b)))$.

The resulting sample of statistics $(t(F_n^*(1)) \ldots t(F_n^*(B)))$ provides an empirical version of $P_{Fn}^{(n)}$, say $P_{B,Fn}^{(n)}$, and we use its quantiles:

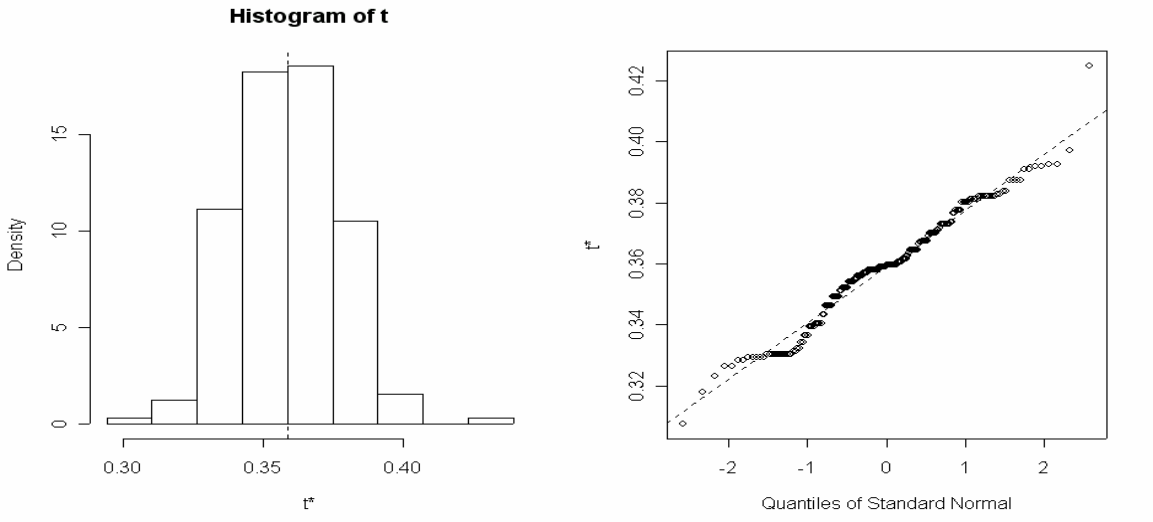
$$\mathrm{CI}_{\mathrm{boot}}(1-2\alpha) = [\mathrm{q}(P_{B,F_n}^{(n)};\alpha), \mathrm{q}(P_{B,F_n}^{(n)};1-\alpha)]$$

Depending on how complicated the statistic is, B in the range of 200-300 already gives a useful percentile CI. For more sophisticated bootstrap-based CIs one may need many more (1000).

Let us compute the (non-parametric) bootstrap-based percentile CI with coverage 95% for the median of y = log-length ratio in chicken_toy. Recall n=100, and we use B=200.

```
> #a special function for CIs takes as input previous output of boot():

> bootCI_loglenrat <- boot.ci(boot_loglenrat,conf=0.95,type="perc")
> bootCI_loglenrat
```

```
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 200 bootstrap replicates

CALL :
boot.ci(boot.out = boot_loglenrat, conf = 0.95, type = "perc")

Intervals :
Level     Percentile
95%   ( 0.3264,  0.3918 )
Calculations and Intervals on Original Scale
Some percentile intervals may be unstable            #use larger B!

> plot(boot loglenrat)              #gives graphical output below
```



Perturbations:

The bootstrap produces B values for the statistic $t(F_n)$, i.e. an empirical proxi for its sampling distribution $P_F^{(n)}$. The standard deviation and quantiles can then be used to approximate standard error and produce confidence intervals, because resampling the data with replacement gives us a way to simulate the sampling variability of the statistic.

What would happen if we produced B values (with their standard deviation and quantiles) resampling the data without replacement? If we took the size of the (re)samples to be n, we would keep re-generating the data set in its entirety, and thus the same statistic value $t(F_n)$. But we could take size $N = f\,n$ (a fraction, say $f = 0.8$). This is equivalent to deleting (1-f) of the data at random.

Resampling without replacement allows us to simulate the distribution of $t(F_n)$ under a special type of perturbation – namely, random deletions, and thus to address **stability** questions.

To these perturbations in R, you can look for ready-to-use functions available on the web (as individual functions or in packages), or you can write your own function. In doing so, you will need to use the function:

```
> #sample(x, size, replace = FALSE, prob = NULL)
> #A deletion perturbation of x is a sample of size f*length(x)
> #from x, without repl. Need not specify sampling weights (prob)
```

Permutation tests:

Consider for instance

- F = population jointly representing y and x both quantitative, and $t(F)=\rho(y,x)$ the correlation.
- F = population jointly representing y (quantitative) and c=1,2 (categorical, indexed), and $t(F)=\mu_1-\mu_2$ the difference between the means.

In both cases the selected population features are ways of measuring the association between two variables. The values representing no association are 0, and we can imagine testing, for instance:

Ho : $\rho(y,x)=0$ (no linear association) vs Ha : $\rho(y,x)>0$
Ho : $\mu_1-\mu_2=0$ (no location effects of the class) vs Ha : $\mu_1-\mu_2>0$

We surely have:
y indep x ➔ Ho : $\rho(y,x)=0$
y indep c ➔ Ho : $\mu_1-\mu_2=0$

Permutation tests exploit the fact that we can simulate independence, and thus, a fortiori, null hypothesis concerning these types of features. Let us consider the difference of means problem:

y	c
y_1	1
.	.
.	.
.	.
y_{n1}	1
y_{n1+1}	2
.	.
.	.
.	.
y_{n1+n2}	2

(sub) sample of size n_1 from the c=1 subpopulation

(sub) sample of size n_2 from the c=2 subpopulation ($n_1+ n_2= n$)

… often called 2-sample problem.

The "plug-in" statistic for the difference of means is $d = \bar{y}_1 - \bar{y}_2$. Keep the y-column fixed, and

For b=1…B

1. Create a random permutation of the c-labels in the second column (equivalent to drawing n times from them *without* replacement). This gives (c_1*(b)… c_n*(b))

2. Using the permuted labels, recompute the difference between the means
   $d^*(b) = \bar{y}_1^*(b) - \bar{y}_2^*(b)$

Compute the permutation-based empirical p-value as

$$p(d_{obs}) = \frac{1}{B}\#(d^*(b) \geq d_{obs})$$

Random permutations simulate sampling from a population in which y and c are left unchanged marginally, but any association existing between them is broken. We are simulating the **null distribution** of the test statistic.

At the sample level, all the marginal features of y (overall mean, sd, etc) and c (frequencies) are preserved, because the y and c numbers stay the same, but the connection is "scrambled" – thus the means within the groups change.

Note that only one of the data columns is permuted (could permute both, but it would not help).

To implement permutation tests in R, you can look for ready-to-use functions available on the web (as individual functions or in packages), or you can write your own function. In doing so, you will need to use the function:

```
> #sample(x, size, replace = FALSE, prob = NULL)
> #A random permutation of x is a sample of size length(x)
> #from x, without repl. Need not specify sampling weights (prob)
```