Week 6 Notes Part 1 Resampling

Code

Author: Brendan Gongol

Last update: 07 January, 2023

1 Overview

In this Note we discuss non-parametric resampling. First we briefly describe the approaches and their rationales, then we consider permutation-based method and bootstrap in more detail and run a few numerical examples in R.

2 Resampling Procedures

We have being dealing with resampling extensively in previous weeks, so the concept should not be completely new by now. The most generic and basic idea behind resampling is that one is performing some kind of brute-force, data-driven simulation, putting together additional examples, and trying to use them in order to assess some properties of the actual sample at hand. There are different ways of implementing this program, and there are different situations, in which such approaches can be justified.

The technique we have been utilizing so far was centered on resampling from the underlying distribution (so we used `rnorm`, `rexp`, etc.). Many of the examples we have considered had purely educational purpose: for instance we were trying to run resampling in order to confirm and better understand an existing analytical procedure such as t-test. In real applications, this approach is still useful in at least two cases:

- The underlying distribution is known, or can be expected to be reasonably close to a known one, but we are studying some complex statistics, for which no analytical expressions are available; in this case we can resample from the known underlying distribution and assess, for instance, the significance of a particular observation (in the same way as we did before: how extreme our observation is compared to a set of simulated "observations" obtained through resampling under the null hypothesis).
- The underlying analytical distribution is not known, but we have extensive (historical) data, i.e. empirical distribution is available. For instance, we have performed a large scale study over the years and thus have the empirical distribution(s) of the quantities of interest. Let's assume that those distributions are very unique and not well approximated by any of standard analytical ones (which is not a very common case, by the way). Now if we run some follow-up experiment and draw a new sample, we can assess its significance by running a brute force resampling simulation against such large historical dataset that represents the empirical distribution.

There are other resampling approaches that can be applied in different situations. The most important and common ones are:

- **Permutation**. In this approach we make no assumptions about the underlying distribution at all. As the name suggests, we randomly reshuffle (permute) the data at hand and assess the resulting distribution of the statistics of interest. This also means that this method is applicable only when the order of observations matters. Indeed, if all we are concerned about is the sample mean, reshuffling the observations will not do much good: every sample generated in this way will have the same mean. What we can test using permutation, however, is association.
- **Bootstrap**. Randomly resample data from the observed data, with replacement, preserving all the structure of the data (group associations etc). Since smaller samples provide less statistical power to detect any particular effect, we usually want to keep the size of the bootstrapped samples equal or at least close to the original one, so that the impact of decreased statistical power is minimized. The main source of sample-to-sample variation in this type of resampling approach is thus due to sampling with replacement. This method is best suited for assessing the stability of the results: if the result we have obtained from the original real-life sample hinges on a couple of outliers, bootstrapping should be able to correct for that. Bootstrap is often used in conjunction with confidence intervals, but any statistics can be assessed.
- **Cross-validation**. A related approach, with the main purpose being the assessment of the predictive power of a model rather than the significance of the model per se. Indeed, it is possible (much more often than you think) to overfit a model to a particular realization of data we have at hand. Then, when we obtain new set of cases and try using our model in order to predict something for those new, yet unseen data (e.g. predict response to a drug in a new group of patients, based on the model we fitted using some preliminary trial data), the model may fail miserably. The reason is that what this model describes can be, to a large extent, accidental idiosyncratic (particular to the individual) "dependences" present due to random chance and sampling error (let alone outliers due to some failed measurements) in the original data the model was trained on. In order to better assess the predictive power of the model, many cross-validation techniques were developed, and we will discuss them in more detail later. It is worth mentioning here though, that such techniques are also resampling-based; for instance we can set aside some part of data, e.g. randomly selected 1/n-th of the observations, train the model on the rest of data, then test the prediction accuracy on this saved, "yet unseen" part of the data; then repeat multiple times.

3 Permutations

uses sample() without replacement - access statistical significance - H_0 testing, comparing group

Let us consider as an example the expression levels of genes across multiple patients (think of our ALL dataset). Gene expression is a complex biological process with extensive regulation and great deal of inter-dependencies. Groups of genes tend to be "co-expressed" (exhibit high degree of correlation from patient to patient: when gene A goes up/down, gene B goes up/down as well), which may reflect interesting underlying biology: such co-expressed genes can be involved in the same biological process and/or be regulated by the same transcription factor(s). In order to quantify the degree of co-expression, we can quickly calculate pairwise correlations between the gene expression profiles across the patients. But does correlation coefficient of 0.85 truly signify co-expression, or it can be observed by random chance, especially when we test so many different pairs of genes (multiple testing is the other subject of this week's material!)?

In order to answer this question, among other things, we can simply independently reshuffle expression levels of each gene across all patients, and recompute the correlation coefficients. We can use a set of correlation coefficients across all genes or repeat this procedure multiple times per gene and thus obtain the

distribution(s) of correlation coefficients possible in this particular dataset under the assumption of no association (because independent reshuffling clearly destroys any association even if it was originally present). Comparing our originally observed correlation coefficient(s) to the resampled distribution, we can get an idea of how significant it is (i.e. how much the specific order of observations in the real data matters). This might be not the perfect answer, but in many situations it can be the only answer we can find, given the limited data.

Another example we are going to look into is the distributions of ages in ALL dataset. We have been looking into this example already, and as you might remember we were examiningwhether there is a significant difference between the (average) ages of male and female patients. In principle, this sounds like a perfect setup for a t-test, except that we are not sure that the underlying distribution the samples are drawn from is normal. We could also run a non-parametric analogue of a t-test (aka rank-sum test, Wilcoxon test, or Mann-Whitney U□test), and we will discuss this test in the coming weeks. But for now let us solve the same problem using permutations.

Our problem is quite simple: we have two samples, we have calculated the difference between their means, and we want to assess the significance of that difference somehow (is that difference large enough so that it is unlikely to be obtained by random chance?). As we have already discussed in relation to the linear models and ANOVA, this problem can beformulated as association between the two variables: age ~ sex, where sex is a categorical variable with two levels, M/F, and we can even fit a linear model (but linear model on categorical independent variable is simply a one-way ANOVA, and since there are only two labels, M/F, in our variable, the result will be the same as the t-test). When we draw patients from the population, we draw an age value from some (unknown) distribution of ages of people suffering from acute lymphoblastic leukaemia, and at the same time we also sample the value of the patient's sex. We don't know what either the age or the sex of the next patient is going to be, so both are random variables. The presence of association would mean, of course, that e.g. when we draw M, the age tends to be higher (or vice versa) in a statistically significant way.

nicely written

Our null hypothesis states, as always, that there is no effect, i.e. age and sex are completely unrelated random variables, and whatever difference we observe in our measured data, it is simply due to chance (random sampling effect). Let us model the effect of the random sampling simply by reshuffling the data at hand (e.g. patient ages, or we could reshuffle the vector of patient's sex, it does not matter). Indeed, if there is no association whatsoever, we could observe the same set of ages randomly reassigned among the patients (with their M/F attributes), or vice versa, the M/F attribute could be observed in any randomized order among the patients with their given ages. We can repeat this reshuffling procedure multiple times and in each resampling we calculate the difference between the means of ages in males and in females. What we arrive to is another approximation to the expected distribution of the difference between mean male and female ages, under the assumption of no association (i.e. underlying means are the same), in other words, under the null.

If the difference in the real sample is "large enough" compared to the null distribution obtained through such reshuffling, then it is "significant" (we have seen this logic in all the resampling procedures we tried so far!). Let us try running this in R:

Hide

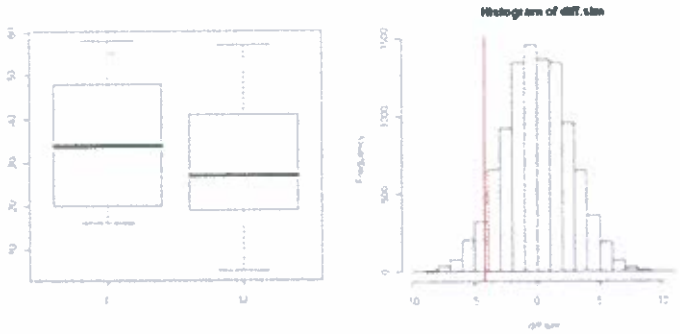
```
library(ALL); data(ALL)
x.age <- pData(ALL)$age                    # extract observations for age and sex
x.sex <- pData(ALL)$sex
x.keep <- !is.na(x.age)&!is.na(x.sex)
x.age <- x.age[x.keep]                     # remove NA's from the data
x.sex <- x.sex[x.keep]
diff.ori <- mean(x.age[x.sex=="M"]) -
 mean(x.age[x.sex=="F"])                   # difference in original sample
diff.sim <- numeric()
for ( i in 1:10000 ) {
x.tmp <- sample(x.age)                     # in this call, sample() simply reshuffles
diff.sim[i] <- mean(x.tmp[x.sex=="M"]) -
mean(x.tmp[x.sex=="F"])                    # difference in reshuffled
 }
sum(abs(diff.sim) >= abs(diff.ori)) / 10000 # brute-force p-value
```

```
## [1] 0.1049
```

Hide

```
t.test(x.age[x.sex=="M"],
x.age[x.sex=="F"],var.equal=T)$p.value
```

```
## [1] 0.1064217
```



The left figure shows the boxplot of male and female patients' ages (same plot as we have seen before, just reproduced here). The right figure shows the distribution of the differences that could be obtained from the same set of ages with just different random reassignments of M/F to each age value. Red line shows where the difference observed in the original, real sample is located. As we can see, this difference does not look too significant. As you can see in the code fragment, the brute-force p-value obtained using the permutation approach is ~0.1. Note that t-test returns a similar value, confirming our previous observation that t-test is actually sufficiently robust and often gives reasonable estimates (especially with larger samples) even when the underlying assumptions (normality of the distribution) are broken.

4 Bootstrap

As we have briefly described above, bootstrapping approach is based on resampling with replacement. It is thus similar to permutations in that it represents a case of non-parametric method, relying only on the data at hand and making no assumptions whatsoever about the underlying distribution. It is, at the same time,

very different from permutation-based approaches in that instead of changing the order of the data (which would have no effect on many statistics such as e.g. the mean, sd, or the confidence interval of the mean), it essentially assesses significance through measuring the effect of outliers (when resampling with replacement, we are effectively asking: "what would happen if a few data points were not in our dataset and were instead replaced by few other data points from the same set").

Let us illustrate bootstrap with a simple example:

sample(x) ← default sample size is the same as length(x).

Hide

```
# random sample from normal; this is our "measured data":
x <- rnorm(100)
v <- numeric(10000)        # reserve the vector of length 10K
for ( n in 1:10000 ) {     # resamplings: bootstrap 10K times
# new sample x1: resampling with replacement; the size of the sample,
# and thus its statistical power, is going to be the same, but due
# to replacement a few randomly chosen points from the original data
# will be dropped and replaced by additional copies of other points
 x1 <- sample(x,replace=T)
 v[n] <- mean(x1)          # calculate the mean of bootstrapped sample
}
sd(v)                      # standard deviation of our resampled mean estimates (SEM)
```

```
## [1] 0.1110801
```

Hide

```
sd(x)/10
```

$\leftarrow \frac{s}{\sqrt{n}}$ $\sqrt{100} = 10$.

```
## [1] 0.1117812
```

In this code fragment, we first generate a sample of size 100 from the normal distribution. In our experiment, this vector represents the data we have "measured". Suppose we do not know anything about the underlying distribution and want to estimate the error of the mean in our sample. In other words, we would like to estimate the distribution of the sample mean as a random variable. Strictly speaking, even lacking any knowledge about the underlying distribution, we still know in advance, from theoretical considerations, what the best estimator is going to be: $s/\sqrt{n}$, where s is standard deviation (sample estimator); so the main purpose of this exercise is to demonstrate how bootstrap works. We use this exercise to test our new method, precisely because we know in advance what the correct answer is, so we probably would not need to run the code for this specific statistic in real life. In our particular case, since we also know the sd of the underlying distribution, we know what all our estimates should converge to the "ultimately" correct answer \$/=0.1 \$ for $\sigma = 1$ and $n = 100$.

In the code shown, we perform 10,000 bootstrapping iterations. In every such iteration, we generate a new sample x1 by resampling x with replacement (i.e. some elements of x do not get selected, while others get selected multiple times, for the total of 100 elements). We calculate and save the mean of each such bootstrapped sample. After all 10,000 resamplings are done, we have an (empirical) distribution of the sample means stored in vector v. The standard deviation of those resampled sample means, as shown in the code fragment, is very close to the true underlying value (0.1) of the SEM at n=100. Moreover, if we use a sample estimator $s/\sqrt{n}$, then, as the last line in the code shows, the difference between this estimator

and the one obtained through bootstrap is of the order of 0.1%. This comparison is fairer, since the bootstrap does not know anything about the underlying distribution and as far as we limit ourselves to the available data at hand, it can hardly be expected to perform better than specifically known, unbiased estimator!

Note that permutations (reshuffling without replacement) would not do much good in this case: no matter how we reshuffle the original vector x, the mean would be exactly the sameWhat is really important here is that our last simulation looks very similar to what we did in earlier weeks: resample 10000 new, independent samples from the normal, calculate the mean for each of them, then calculate the standard deviation of that mean. You may remember from our studies that the sd of the mean (i.e. SEM) was indeed $s/\sqrt{n}$, as expected, and this was how we demonstrated SEM in action. The only place where the code shown above differs from our old simulation is that now the new, resampled sample x1 is notdrawn from the underlying distribution (we are not making any assumptions, whatsoever, about that underlying distribution!!), but is instead re-drawn, with replacement, from the original data at hand. Astonishingly, we were still able to get the correct estimate for the SEM by performing bootstrap resampling from the single available sample, without any assumptions! In retrospect, what we did in previous weeks was a self-serving and more educational procedure: it is much less surprising that resampling new, independent realizations from the underlying distribution, which we know a priori, gives the correct result: it does so pretty much by definition; in real life, however, we rarely have the luxury of drawing multiple completely new examples of the data (and when we do know the underlying distribution, there is often a ready-to use analytical test, so resampling is not needed)! It is when we do not want to make any assumptions and want to derive as much as we can from the data at hand only, the bootstrap really shines.

R has built-in functions for running bootstrap (available from library 'boot'). Their interface is a little tricky (but also flexible), so let us peruse our example of male vs female patient age difference in the ALL dataset in order to see how these functions work:

Hide

```
library(boot)
MF.age.diff <- function(x,i) {
 age.m <- x[i,"age"][x[i,"sex"]=="M"]
 age.f <- x[i,"age"][x[i,"sex"]=="F"]
 mean(age.m)-mean(age.f)
}
x.df.tmp <- data.frame(age=x.age,sex=x.sex)
boot.res <- boot(x.df.tmp,
 statistic=MF.age.diff,10000,strata=x.sex)
boot.ci(boot.res,type="norm")$normal
```

```
##      conf
## [1,] 0.95 -9.33895 0.9530779
```

Hide

```
t.test(x.age[x.sex=="M"],
 x.age[x.sex=="F"],var.equal=T)$conf.int
```

```
## [1] -9.4021924  0.9207109
## attr(,"conf.level")
## [1] 0.95
```

The function 'boot' provides the core functionality of bootstrapping. We will be studying only its simplest use here. This function takes data as the first argument, and for that argument we pass a dataframe that contains two columns: patient's age and sex. The 'statistic' parameter is a function (just like one of the parameters of `apply()` is a function). The contract for this function (in the default mode implemented in `boot()`) is as follows: the function should take at least two arguments; the first argument is the data, the second argument is the selected indices. The function is expected to return the statistics of interest. Next argument is the number of bootstrap iterations (10,000 in our case). The `boot()` function works as follows: it performs the requested number of bootstrap iterations; in each iteration it chooses (randomly, with replacement) the indices of the data values to be selected. Then it passes full original data (same data we passed to the `boot()` itself) and those selection indices to its function argument and collects the output (statistics) computed by that function.

Our function `MF.age.diff()` implements the contract so that it could be used with `boot()` : it takes the (whole) dataset as its first argument, and the selection indices as the second argument. Inside the function body we take the requested bootstrap of data according to the indices passed from `boot()` (indeed, when we are requested to use the data points with indices in the vector `i` , the "full data" in that iteration is `x[i,]`), , and compute the difference between the means of male and female ages in that particular bootstrapped sample; the latter difference is the statistics we want to analyze by boostrap, so we simply return it. Note that in our case we also ask boot to stratify the data by sex and perform resampling separately in M/F groups (the `strata=` argument). This way we are requesting that male and female ages are bootstrapped (resampled with replacement) separately, so that the total numbers of males and females in each bootstrapped sample remains the same (without the strata argument, boot would do unrestricted resampling on the whole dataset, so it could drop, say, three males and replace these data points with three females – this is not necessarily gravely incorrect, you just need to appreciate the fact that this is a slightly different variant of the resampling strategy).

The `boot()` function returns an object holding multiple data elements (see the docs). In particular, the element `t0` contains the value of statistics the function we passed to boot() calculated on the whole original dataset, and the element t in the returned object contains the value of the statistics function for each resampling (hence the number of elements in t is equal to the number of iterations performed, 10,000 in our case).

The next function we call in the code fragment above is `boot.ci()` . The latter companion function simply takes the object returned by `boot()` (i.e. the results of bootstrapping) as its argument and calculates confidence interval of the statistics (based on the distribution of bootstrapped values passed in the `boot.res$t`). While the precise details can be a little over our heads, there no magic here. Remember how the t-test works. We have the sample mean. We assume normality, so we know how the sample mean, as a random variable, is distributed: a normal itself with $sd = \sigma/\sqrt{n}$. From that, we can immediately calculate the interval that contains, e.g., 95% of the mass of the distribution (95% confidence interval). When we do not know the true sd of the underlying distribution, we replace it with the sample sd (it's the best we can do!), and in this case the t-distribution is the correct one, but this is just smart fine tuning in order to get more accurate result. In any case, in order to calculate the confidence interval all the ingredients we need are the value of the statistic (e.g. sample mean) and its distribution. This is precisely what we pass to `boot.ci()` : the results of bootstrap that, as described above, contain both these ingredients, so there is no wonder we

can get an estimate for the confidence interval of the statistic, somehow. The beauty (or the problem, if you wish) is that in this approach we made no assumptions, and the distribution of the sample statistic of interest is an empirical one, an approximation calculated through bootstrap resamplings.

Since we are dealing with approximate, empirical distribution, the function `boot.ci()` can numerically calculate few different types of confidence intervals based on different further assumptions (yes, we do have to make some, at some point) and different numerical interpolations. In our example we asked for the confidence interval based on the normal approximation (you can check the other flavors of the confidence interval calculations and observe that they all give very similar results in this case). Theory behind types of confidence intervals and various numerical interpolations is beyond the scope of this course; it suffices to say that we can get a confidence interval, and that the different approximations are going to be very close to one another in most practical cases.

The output of the `boot.ci()` in our code example shows that the 95% confidence interval of the difference between mean ages of male and female patients, as assessed by bootstrapping, is [-9.32, 0.88]. It contains zero, and hence we again observe no significant differences between mean ages. Lastly, we compute the confidence interval using t-test function and, as we can see, it is indeed very close to our boostrapped estimate computed using only the sample itself and nothing else (no assumptions!).

emperical distribution ← data from observed points.