CRAN has accepted ‘pdqr’ package. Here are important examples of how it can be used to describe and evaluate statistical uncertainty.

**Prologue**

We will need the following setup:

library(pdqr)

library(magrittr)

# For the sake of reproducibility

set.seed(20191111)

**Statistical uncertainty**

**General description**

Statistical estimation usually has the following setup. There is a sample (*observed, usually randomly chosen, set* of values of measurable quantities) from some general population (*whole set* of values of the same measurable quantities). *We need to make conclusions about the general population based on a sample*. This is done by computing summary values (called **statistics**) of a sample, and making reasonable assumptions (with process usually called **inference**) about how these values are close to values that *potentially can be computed based on whole general population*. Thus, summary value based on a sample (**sample statistic**) is an *estimation* of potential summary value based on a general population (true value).

How can we make inference about quality of this estimation? This question itself describes **statistical uncertainty** and can be unfolded into a deep philosophical question about probability, nature, and life in general. Basically, the answer depends on assumptions about the relation between sample, general population, and statistic.

For me, the most beautiful inferential approach is [**bootstrap**](https://en.wikipedia.org/wiki/Bootstrapping_(statistics)). It has the following key assumption: *process of producing samples from general population can be simulated by doing random sampling with replacement from present sample*. In other words, we agree (and practice often agrees with us) that random sampling with replacement from current sample (sometimes called *bootstrap sampling*) has a “close enough” behavior to the “true nature” of how initial sample was created. Numerical estimation of “how close” is also an interesting problem, but it is a more complicated topic.

**Computation with pdqr**

Natural way of computing bootstrap quantities is straightforward: produce \(B\) random bootstrap samples, for each one compute value of statistic under question, and summarize *sample of statistic values* with numerical quantity (usually with some center and spread values).

There are many ways of performing bootstrap in R, like boot::boot(), rsample::bootstraps(), and others. In turn, ‘pdqr’ offers its own way of describing and doing bootstrap inference for one-dimensional numeric sample(s):

* **Create** a random variable (in the form of pdqr-function with new\_\*() family) based on initial sample. This random variable already describes a general population with “bootstrap assumption”: it will produce values based on initial sample. Type of this variable determines the type of bootstrap:
  + Type "discrete" describes ordinary bootstrap. Only values from initial sample can be produced.
  + Type "continuous" describes smooth bootstrap. Initial sample is smoothed by doing kernel density estimation with density() function and random variable produces values from distribution with that density.
* **Transform** created random variable into one that produces statistic values obtained with bootstrap. Sometimes this can be done with basic mathematical operations like +, min, etc. But usually this is done with form\_estimate() function: it creates many (10000 by default) bootstrap samples, for each computes statistic value, and creates its own random variable in the form of pdqr-function (class and type are preserved from supplied random variable, but this can be adjusted). It needs at least three arguments:
  + f: pdqr-function representing random variable. In described setup it is created as a result of “Create” step.
  + stat: statistic function that accepts numeric vector of size sample\_size and returns single numeric or logical output.
  + sample\_size: Size of a sample that each bootstrap draw should produce. In described setup it *should be equal* to number of elements in initial sample.
* **Summarize** distribution of statistic. Usually this is point measure of center or spread, or interval.

**Example 1: single numerical estimate**

Mean value of ‘mpg’ variable in mtcars dataset is 20.090625. However, having in mind statistical uncertainty, we can ask how precise is this estimation? This can, and should, be reformulated in the following question: **if we repeat sampling sets of 32 cars from general population of all cars, how close their ‘mpg’ sample means will be to each other?** This can be answered by computing bootstrap distribution of sample means:

# Using ordinary bootstrap

d\_mpg\_dis\_mean <- mtcars$mpg %>%

new\_d(type = "discrete") %>%

form\_estimate(stat = mean, sample\_size = nrow(mtcars))

# Spread of this bootstrap distribution describes the precision of estimation:

# bigger values indicate lower precision

summ\_sd(d\_mpg\_dis\_mean)

## [1] 1.04067

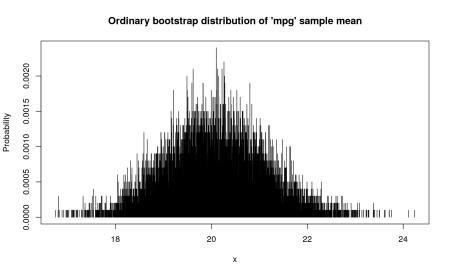
# This discrete distribution has the following d-function

plot(

d\_mpg\_dis\_mean,

main = "Ordinary bootstrap distribution of 'mpg' sample mean"

)



If modeling assumption about continuous nature of ‘mpg’ variable is reasonable (which it seems so), you can use “smooth bootstrap” by changing type of initial pdqr-function:

# Using smooth bootstrap with `type = "continuous"`

d\_mpg\_con\_mean <- mtcars$mpg %>%

new\_d(type = "continuous") %>%

form\_estimate(stat = mean, sample\_size = nrow(mtcars))

# Spread is higher in this case because kernel density estimation with

# `density()` function extends support during creation of pdqr-function on the

# bootstrap step

summ\_sd(d\_mpg\_con\_mean)

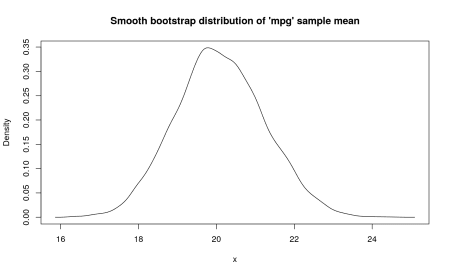
## [1] 1.153957

plot(

d\_mpg\_con\_mean,

main = "Smooth bootstrap distribution of 'mpg' sample mean"

)



One can also do ordinary bootstrap but represent bootstrap distribution of sample mean with continuous random variable:

# Using ordinary bootstrap, but treating sample mean as continuous

d\_mpg\_con\_mean\_2 <- mtcars$mpg %>%

new\_d(type = "discrete") %>%

form\_estimate(

stat = mean, sample\_size = nrow(mtcars),

# Create continuous pdqr-function from bootstrap sample means

args\_new = list(type = "continuous")

)

summ\_sd(d\_mpg\_con\_mean\_2)

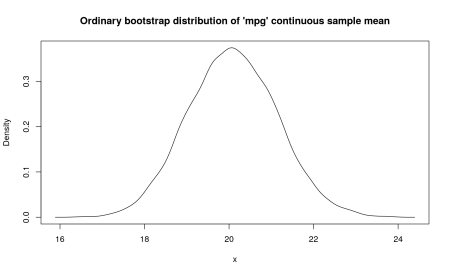
## [1] 1.063524

plot(

d\_mpg\_con\_mean\_2,

main = "Ordinary bootstrap distribution of 'mpg' continuous sample mean"

)



In this case, sample mean has standard deviation from 1.04067 to 1.1539572 (depends on assumptions about data generating process).

**Example 2: single logical estimate**

Share of 4-cylinder cars in mtcars is equal to 0.34375. However, it might happen that we don’t care about actual value, but only if it is bigger 0.3 or not. In present data it is bigger, but how sure we can be about that? In other words: **if we repeat sampling sets of 32 cars from general population of all cars, which part of it will have share of 4-cylinder cars bigger than 0.3?**. Here is the way of computing that with ‘pdqr’:

# If statistic returns logical value (indicating presence of some feature in

# sample), output estimate pdqr-function is "boolean": "discrete" type function

# with elements being exactly 0 (indicating `FALSE`) and 1 (indicating `TRUE`).

d\_cyl\_lgl <- mtcars$cyl %>%

new\_d(type = "discrete") %>%

form\_estimate(

stat = function(x) {mean(x == 4) > 0.3},

sample\_size = nrow(mtcars)

)

d\_cyl\_lgl

## Probability mass function of discrete type

## Support: [0, 1] (2 elements, probability of 1: 0.7113)

# To extract certain probability from boolean pdqr-function, use

# `summ\_prob\_\*()` functions

summ\_prob\_true(d\_cyl\_lgl)

## [1] 0.7113

summ\_prob\_false(d\_cyl\_lgl)

## [1] 0.2887

In this case, estimated probability that share of 4-cylinder cars in general population is more than 0.3 is 0.7113.

**Example 3: comparison of estimates**

In mtcars there are 19 cars with automatic transmission (‘am’ variable is 0) and 13 with manual (‘am’ variable is 1). We might be concerned with the following question: **are cars with automatic transmission heavier than cars with manual transmission?** This is an example of question where reformulating is very crucial, because it leads to completely different methodologies. Basically, it is all about dealing with statistical uncertainty and how to measure that one numerical set is bigger than the other.

First, rather verbose, way of expanding this question is this one: **if we randomly choose a car with automatic transmission (uniformly on set of all cars with automatic transmission) and a car with manual (uniformly on set of all cars with manual transmission), what is the probability that weight of the first one is bigger than the second one?**. With ‘pdqr’ this can be computed straightforwardly by comparing two random variables:

# Seems reasonable to treat weight as continuous random variable. Note that this

# means use of kernel density estimation, which can lead to random variable that

# returns negative values. As weight can be only positive, it is a good idea to

# ensure that. Package 'pdqr' has `form\_resupport()` function for that.

d\_wt\_am0 <- mtcars$wt[mtcars$am == 0] %>%

new\_d(type = "continuous") %>%

# Ensure that returned values are only positive

form\_resupport(c(0, NA))

d\_wt\_am1 <- mtcars$wt[mtcars$am == 1] %>%

new\_d(type = "continuous") %>%

form\_resupport(c(0, NA))

# Comparing two pdqr-functions with `>=` results into boolean pdqr-function

summ\_prob\_true(d\_wt\_am0 >= d\_wt\_am1)

## [1] 0.9209063

So in this case the answer is that probability of “automatic” cars being heavier than “manual” ones is around 0.921.

Second way of understanding question about comparing is the following: **is average weight of “automatic” cars bigger than of “manual”?**. This type of questions are more widespread in statistical practice. Having to deal with statistical uncertainty, this should be reformulated: **if we repeat sampling (in parallel pairs) sets of 19 “automatic” cars and of 13 “manual” cars, which part of the set pairs will have mean weight of “automatic” cars bigger?** This question implies creating bootstrap distribution of sample means for “automatic” and “manual” cars with the following comparing:

d\_wt\_am0\_mean <- d\_wt\_am0 %>%

form\_estimate(stat = mean, sample\_size = sum(mtcars$am == 0)) %>%

# Ensure "positiveness" of random variable

form\_resupport(c(0, NA))

d\_wt\_am1\_mean <- d\_wt\_am1 %>%

form\_estimate(stat = mean, sample\_size = sum(mtcars$am == 1)) %>%

form\_resupport(c(0, NA))

# Comparing two random variables representing sample means

summ\_prob\_true(d\_wt\_am0\_mean >= d\_wt\_am1\_mean)

## [1] 1

So in this case the answer is that probability of “automatic” cars being heavier than “manual” ones is 1.

Computed results can have decisively different outcomes. If researcher sets a standard 0.95 rule, first variant would imply that conclusion ‘“automatic” cars are heavier than “manual”’ isn’t significant, while the second would imply otherwise.

**Epilogue**

* Basic knowledge about statistical uncertainty is crucial to understand the process of statistical inference.
* One of the most popular methodologies for doing statistical inference is bootstrap. There are at least two kinds of it: ordinary and smooth.

sessionInfo()

sessionInfo()

## R version 3.6.1 (2019-07-05)

## Platform: x86\_64-pc-linux-gnu (64-bit)

## Running under: Ubuntu 18.04.3 LTS

##

## Matrix products: default

## BLAS: /usr/lib/x86\_64-linux-gnu/openblas/libblas.so.3

## LAPACK: /usr/lib/x86\_64-linux-gnu/[libopenblasp-r0.2.20.so](http://libopenblasp-r0.2.20.so)

##

## locale:

## [1] LC\_CTYPE=ru\_UA.UTF-8 LC\_NUMERIC=C

## [3] LC\_TIME=ru\_UA.UTF-8 LC\_COLLATE=ru\_UA.UTF-8

## [5] LC\_MONETARY=ru\_UA.UTF-8 LC\_MESSAGES=ru\_UA.UTF-8

## [7] LC\_PAPER=ru\_UA.UTF-8 LC\_NAME=C

## [9] LC\_ADDRESS=C LC\_TELEPHONE=C

## [11] LC\_MEASUREMENT=ru\_UA.UTF-8 LC\_IDENTIFICATION=C

##

## attached base packages:

## [1] stats graphics grDevices utils datasets methods base

##

## other attached packages:

## [1] magrittr\_1.5 pdqr\_0.2.0

##

## loaded via a namespace (and not attached):

## [1] Rcpp\_1.0.2 bookdown\_0.13 crayon\_1.3.4 digest\_0.6.21

## [5] evaluate\_0.14 blogdown\_0.15 pillar\_1.4.2 rlang\_0.4.0

## [9] stringi\_1.4.3 rmarkdown\_1.15 tools\_3.6.1 stringr\_1.4.0

## [13] xfun\_0.9 yaml\_2.2.0 compiler\_3.6.1 htmltools\_0.3.6

## [17] knitr\_1.25