R hacking
aneb
vybrané heuristiky data processingu a efektivního kódu

—

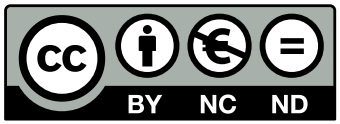
Statistické dýchánky na VŠE

Lubomír Štěpánek[1, 2]

[1]Oddělení biomedicínské statistiky & výpočetní techniky
Ústav biofyziky a informatiky
1. lékařská fakulta
Univerzita Karlova v Praze

ČVUT
FBMI

[2]Katedra biomedicínské informatiky
Fakulta biomedicínského inženýrství
České vysoké učení technické v Praze



Obsah

Skript k přednášce

- skript k přednášce je zde

https://github.com/LStepanek/r_hacking/blob/master/__script__.R

Úvod

- processing dat je vždy nutné individualizovat pro konkrétní data
- pravděpodobně neexistuje jednotný a nejlepší přístup pro obecná data
- našimi největšími spojenci v této bitvě jsou

```
assign()
get()
apply(); lapply()
eval(parse(text = "..."))
do.call()
grepl(); gsub()

# pro začátek nepohrdneme ani
for(){}
```

- a mnozí další

Hinty na začátek

- obecně je lepší používat generickou funkci než její wrapper
  - je lepší upřednostnit `read.table()` před `read.csv()`, `read.csv2()`, `read.delim()` či `read.delim2()`
  - zejména pro velké datasety

Nahrávání většího množství datasetů současně

- mějme datasety v jedné složce (např. ve složce `vstupy`)
- všechny datasety mají stejnou příponu `.suffix`, pak

```
setwd(choose.dir())   # pop-up okno
                      # pro výběr pracovní složky
for(my_filename in dir()){
    assign(
        gsub(
            "\\.suffix$", "", my_filename
        ),
        read.table(
            file = my_filename,
            ...,      # další parametry
            encoding = "UTF-8"
        )
    )
}
```

Import „messy“ dat

- pokud mají data nějakou pravidelnou strukturu, lze zkusit `read.table()`
- jinak je naším přítelem `readLines()` pro načítání volného textu s vhodnou volbou kódování textu
  - protože všechny datové typy jsou vlastně jen text
- další fází je pak různě náročný *processing*

Processing „messy“ dat

- načítání volného textu s kódováním

```
my_html <- readLines(
    con = "http://dychanky.vse.cz/",
    encoding = "UTF-8"
      # v úvahu připadá "latin1", "latin2", "ASCII"
)
```

- náhrada pevných mezer vlastním tagem

```
my_text <- gsub(" ", "M_E_Z_E_R_A",
                my_html)
```

- vymazání všech HTML tagů a entit

```
my_text <- gsub("<.*?>", "", my_text)
my_text <- gsub("&.*?;", "", my_text)
my_text <- gsub("M_E_Z_E_R_A", " ", my_text)
      # nahrazuji nezlomitelnou mezeru zlomitelnou
```

Processing „messy“ dat

- odstranění white strapes

```
for(i in 1:length(my_text)){
  while(grepl("  ", my_text[i])){
    my_text[i] <- gsub("  ", " ", my_text[i])
  }
}
```

- ponechání jen řádků obsahujících text

```
my_text <- my_text[!my_text %in% c("", " ")]
```

- jednoduchá úloha – extrakce emailů

```
gsub(
  "(.*?)([a-zA-Z\\.]+@[a-zA-Z]+\\.[a-zA-Z]+)(.*)"
    ,
  "\\2",
  my_text[grepl("@", my_text)]
)
```

Processing „messy“ dat

- někdy se může hodit odstranění diakritiky

```
my_text <- iconv(my_text,
                 from = "UTF-8",
                 to = "ASCII//TRANSLIT")
```

Hromadné provedení nějaké operace na více datasetech

- chceme např. (prostě proto) zvýšit hodnoty i-tého datasetu (`dataset_`i`.txt`) právě i-krát a poté o i^2, kde $i \in \{1, 2, \ldots, 100\}$

```
for(i in 1:100){
    my_filename <- paste(
        "dataset_",
        paste(
            rep("0", 3 - nchar(as.character(i))),
            collapse = ""
        ),
        i,
        sep = ""
    )

    assign(my_filename,
           get(my_filename) * i + i ^ 2)
}
```

Automatizace leave-one-out regrese

- v datasetu `mtcars` chceme postupně sestavit všechny modely pro vysvětlovanou proměnnou `mpg`, kdy vysvětlujícími proměnnými budou všechny ostatní proměnné vždy až na jednu vynechanou
- výstup vypišme do souboru volným textem

Automatizace leave-one-out regrese

```
sink("leave_one_out_regrese.txt")

for(my_variable in setdiff(colnames(mtcars), "mpg")
    ){
  my_formula <- paste("mpg ~ ",
      paste(setdiff(colnames(mtcars),
            c("mpg", my_variable)), collapse = " +
                ),
      sep = "")
  eval(parse(text = paste(
       "my_lm <- lm(", my_formula, ", data = mtcars
           )",
       sep = "")))
  print(summary(my_lm))
}

sink()
```

Import a export dat z a do MS Excel® (`.xlsx`)

- vhodný je například balíček `openxlsx`
  - má výhodu, že narozdíl od balíčku např. `xlsx` nepotřebuje Java Tool Kit, takže dokáže najednou nahrát hodně souborů MS Excel®
  - výstup do MS Excel® lze libovolně upravit již v R
- MS Excel® je archivovaný formát
  - zkuste přejmenovat příponu `.xlsx` na `.zip` a rozbalit – výsledkem je složka se spoustou XML souborů)

Import a export dat z a do MS Excel® (.xlsx)

- uložení tabulky do excelového formátu (.xlsx)

```
browseURL(paste(
  "https://raw.githubusercontent.com/LStepanek",
  "17VSADR_Skriptovani_a_analyza_dat_v_jazyce_R",
  "master/export_dat_do_ms_excelu.R", sep = "/"
))
```

- načtení excelové tabulky

```
my_data<- read.xlsx(
  xlsxFile = "moje_tabulka_je_ted_v_excelu.xlsx",
  sheet = 1,    # anebo jméno listu
  colNames = TRUE
)
```

SQL v R

- pomocí balíčku `sqldf`

```
library(sqldf)

sqldf("
  SELECT *
  FROM mtcars
")      # vypisuji všechny sloupce mtcars

sqldf("
  SELECT cyl,
         round(avg(mpg), 2) AS 'mean_mpg'
  FROM mtcars
  GROUP BY cyl
  ORDER BY cyl
")      # vypisuji tabulku počtu válců a průměrných
        # mílí na galon pro každý počet válců
```

Iterování nad funkcemi

- předpokládejme, že chceme pro vektor hodnot `x` zjistit postupně průměr, minimum, maximum, medián, směrodatnou odchylku, varianci a vždy poslední cifru čísla

```
set.seed(1); x <- floor(runif(100) * 100)

for(my_function in c(
    "mean",
    "min",
    "max",
    "median",
    "sd",
    "var",
    function(i) i %% 10
)){
    print(do.call(my_function, list(x)))
}
```

Evangelia

1. Miluj syntax svou!
2. Budeš vektorizovat!
3. Vektory Tobě svěřené nenecháš iterativně růst!
4. Budeš ctít funkce rodiny `apply()` Tobě představené!
5. Ne-`for()`-smilníš!
   - Leda že by ta `for()` smyčka za to fakt stála...
6. Paralelizuj, jsi-li hoden.
7. Mocné umění C++ v R dobře skrývej před nepřítelem!

Miluj syntax svou!

- smart code má svá pravidla, i když se tím rychlost jeho provedení nezvýší
  - na jeden řádek patří maximálně 80 znaků
  - indentace kódu je vhodná, alespoň dvě mezery (lépe čtyři) na úroveň
  - názvy proměnných a funkcí vhodné sjednotit, např. proměnné pomocí `underscore_notace` a funkce pomocí `camelCaps`

Miluj syntax svou!

- ode dneška zapomeňme na adresaci dolarem ($) !!!

```
for(my_variable in colnames(mtcars)){
  print(mean(mtcars$my_variable))
}    # nebude fungovat

for(my_variable in colnames(mtcars)){
  print(mean(mtcars[, my_variable]))
}  # mnohem lepší
```

- (fajnšmejkři dokážou žít i s dolarem)

```
for(my_variable in colnames(mtcars)){
    print(eval(parse(text = paste(
        "mean(mtcars$", my_variable,
        ")", sep = ""
    ))))
}
```

If you are a ~~looser~~ looper, you are on your own!

- vytvořme vektor třetích mocnin čísel 1 až 1000

If you are a ~~looser~~ looper, you are on your own!

- vytvořme vektor třetích mocnin čísel 1 až 1000

```
# I am a looper!
x <- NULL
for(i in 1:1000){x <- c(x, i ^ 3)}

# I am somewhere in the middle
x <- rep(0, 1000)
for(i in 1:1000){x[i] <- i ^ 3}

# I vectorise, anytime, anyhow!
x <- c(1:1000) ^ 3
```

If you are a ~~looser~~ looper, you are on your own!

- vytvořme vektor třetích mocnin čísel 1 až 1000

```
my_start <- Sys.time()
x <- NULL                # I am a looper!
for(i in 1:1000){x <- c(x, i ^ 3)}
my_stop <- Sys.time(); my_stop - my_start  # 0.050s

my_start <- Sys.time()
x <- rep(0, 1000)   # I am somewhere in the middle
for(i in 1:1000){x[i] <- i ^ 3}
my_stop <- Sys.time(); my_stop - my_start  # 0.037s

my_start <- Sys.time()
x <- c(1:1000) ^ 3 # I vectorise, anytime, anyhow!
my_stop <- Sys.time(); my_stop - my_start  # 0.015s
```

Intermezzo

- Najděme pomocí Monte-Carlo integrace velikost určitého integrálu

$$\int_0^1 x^2 \mathrm{d}x.$$

Intermezzo

- Najděme pomocí Monte-Carlo integrace velikost určitého integrálu

$$\int_0^1 x^2 \mathrm{d}x.$$

- zřejmě je

$$\int_0^1 x^2 \mathrm{d}x = \left[\frac{x^3}{3}\right]_0^1 = \frac{1}{3} - 0 = \frac{1}{3}.$$

Intermezzo

- Najděme pomocí Monte-Carlo integrace velikost určitého integrálu

$$\int_0^1 x^2 \mathrm{d}x.$$

```
# I am a looper!
n_of_hits <- 0
for(i in 1:100000){
  if(runif(1) < runif(1) ^ 2){
        n_of_hits <- n_of_hits + 1
  }
}
n_of_hits / 100000
```

Intermezzo

- Najděme pomocí Monte-Carlo integrace velikost určitého integrálu

$$\int_0^1 x^2 \mathrm{d}x.$$

```
# I vectorise, anytime, anyhow!
mean(runif(100000) <= runif(100000) ^ 2)
```

Rodina funkcí `apply()`

- jde o funkce dobře optimalizované tak, že v rámci svého vnitřního kódu „co nejdříve“ volají C++ ekvivalenty R-kové funkce
- díky tomu jsou exekučně rychlé
- nejužitečnější je `apply()` a `lapply()`

Funkce apply()

- vrací vektor výsledků funkce `FUN` nad maticí či datovou tabulkou `X`, kterou čte po řádcích (`MARGIN = 1`), nebo sloupcích (`MARGIN = 2`)
- syntaxe je `apply(X, MARGIN, FUN, ...)`

```r
apply(mtcars, 2, mean)

my_start <- Sys.time()
x <- apply(mtcars, 2, mean)
my_stop <- Sys.time(); my_stop - my_start # 0.019s

my_start <- Sys.time()
x <- NULL
for(i in 1:dim(mtcars)[2]){
  x <- c(x, mean(mtcars[, i]))
  names(x)[length(x)] <- colnames(mtcars)[i]
}
my_stop <- Sys.time(); my_stop - my_start # 0.039s
```

Funkce `lapply()`

- vrací list výsledků funkce `FUN` nad vektore či listem `X`
- syntaxe je `lapply(X, FUN, ...)`
- **skvěle se hodí pro přepis `for()` cyklu do vektorizované podoby!**
- vhodná i pro adresaci v listu

```
set.seed(1)
my_long_list <- lapply(
    sample(c(80:120), 100, TRUE),
    function(x) sample(
        c(50:150), x, replace = TRUE
    )
)    # list vektorů náhodné délky
     # generovaných z náhodných čísel

lapply(my_long_list, "[[", 14)
     # z každého prvku listu (vektoru)
     # vybírám jen jeho 14. prvek
```

Náhrada for cyklu funkcí lapply()

- obě procedury jsou ekvivalentní stran výstupu, `lapply()` je významně rychlejší

```
# for cyklus
x <- NULL
for(i in 1:N){
  x <- c(x, FUN)
}

# lapply
x <- unlist(
  lapply(
    1:N,
    FUN
  )
)
```

Náhrada for cyklu funkcí lapply()

```r
# for cyklus
my_start <- Sys.time()

for_x <- NULL
for(i in 1:100000){for_x <- c(for_x, i ^ 5)}

my_stop <- Sys.time(); my_stop - my_start  # 18.45s

# lapply
my_start <- Sys.time()

lapply_x <- unlist(lapply(
  1:100000,
  function(i) i ^ 5    # koncept anonymní funkce
))

my_stop <- Sys.time(); my_stop - my_start  # 0.10s
```

Princip

- úlohy, které ze provádět nezávisle na sobě, není nutné počítat sériově, ale paralelně
- předpokladem je počítač s procesorem o více jádrech
- v R nejlépe pomocí balíčku `parallel`
- v dalších kapitolách jsou příklady a myšlenky z [2]

Paralelizované ekvivalenty apply() funkcí

```
fibonacci <- function(n){
    if(n <= 2){return(1)}
    if(n >= 3){return(fibonacci(n - 1) +
                      fibonacci(n - 2))}}

detectCores()           # počet jader
cl <- makeCluster(1)    # zapojení clusteru
clusterCall(cl, fibonacci)

my_start <- Sys.time()
invisible(parLapply(cl, 1:40, fibonacci))
stopCluster(cl)
my_stop <- Sys.time(); my_stop - my_start  # 0.05s

my_start <- Sys.time()
invisible(lapply(1:40, fibonacci))
my_stop <- Sys.time(); my_stop - my_start  # > 40s
```

Jazyk C++ v R

- v R nejlépe pomocí balíčku `Rcpp`
- ideou je psaní bottleneckových funkcí syntaxí jazyka C++ v prostředí R

Uživatelem definované funkce v R

```
mean_r <- function(x){
  my_mean <- 0
  for(i in 1:length(x)){
    my_mean= my_mean + x[i] / length(x)
  }
  return(my_mean)
}

set.seed(1)
mean_r(rnorm(1000000))
```

Uživatelem definované funkce v C++

```
cppFunction("
  double mean_cpp(NumericVector x) {
    int i;
    int n = x.size();
    double mean = 0;

    for(i = 0; i < n; i++) {
      mean = mean + x[i] / n;
    }
    return mean;
  }
")

set.seed(1)
mean_cpp(rnorm(1000000))
```

Porovnání rychlosti funkcí mean(), mean_cpp() a mean_r()

```
my_start <- Sys.time()
set.seed(1)
mean(rnorm(1000000))       # 4.690776e-05
my_stop <- Sys.time(); my_stop - my_start  # 0.24s

my_start <- Sys.time()
set.seed(1)
mean_cpp(rnorm(1000000)) # 4.690776e-05
my_stop <- Sys.time(); my_stop - my_start  # 0.25s

my_start <- Sys.time()
set.seed(1)
mean_r(rnorm(1000000))    # 4.690776e-05
my_stop <- Sys.time(); my_stop - my_start  # 1.60s
```

Literatura


Hadley Wickham. *Advanced R*. Boca Raton, FL: CRC Press, 2015. ISBN: 978-1466586963.


Colin Gillespie. *Efficient R programming: a practical guide to smarter programming*. Sebastopol, CA: O'Reilly Media, 2016. ISBN: 978-1491950784.

Děkuji za pozornost!

lubomir.stepanek@lf1.cuni.cz
lubomir.stepanek@fbmi.cvut.cz