Funkcije in degugging (iskanje napak, razhroščevanje)

Lara Lusa

Maj 2019

Funkcije

Funkcije so sestavljene iz 3 komponentov

- body()
- formals()
- environment()

```
#definicija prve funkcije
f_sqrt <- function(x){
  #calculates the squre root of a number
  sqrt(x)
}

body(f_sqrt) #komentarji niso vkljuceni
```

```
## {
##     sqrt(x)
## }
```

```
formals(f_sqrt) #seznam argumentov
```

```
## $x
```

```
environment(f_sqrt) #v katerem okolju se nahaja funkcija
```

```
## <environment: R_GlobalEnv>
```

```
attributes(f_sqrt) #dodatne lastnosti (srcref: source reference)
```

```
## $srcref
## function(x){
##   #calculates the squre root of a number
##   sqrt(x)
## }
```

```
is.function(f_sqrt) #preverimo, ali f_sqrt je funkcija
```

```
## [1] TRUE
```

Napovejte rezultat (lexical scoping - kje iskati)

```
#definicija druge funkcije
f_sqrt2 <- function(){
  x <- 9
  #calculates the squre root of a number
  sqrt(x)
}


#definicija tretje funkcije
```

```r
f_sqrt3 <- function(){
  #calculates the squre root of a number
  sqrt(x)
}
```

```r
x <- 25

f_sqrt()
f_sqrt2()
f_sqrt3()
```

Kako preveriti, ali je funkcija odvisna od zunanjih spremenljivk/funkcij

V prejsnjih primerih, ko se spremeni vrednost x-a v splosnem okolju, spremeni se tudi rezultat funkcije.

Zelo pogosto ima uporabnik x v splosnem okolju in pozabi definirati kaksen od "notranjih" parametrov. R najde x v splosnem okolju, ne javi napake, ki se bo pojavila, ko bo novi uporabnik zelel uporabiti funkcijo oziroma bodo rezultati napacni.

Zato je ponavadi dobro se izogniti zunanjih odvisnosti! Lahko preverimo, ali funkcije imajo kaksno zunanjo odvisnost s funkcijo `codetools::findGlobals`.

Napovejte rezultat - uporaba predefiniranih vrednosti oz. neobveznih parametrov

```r
#definicija cetrte funkcije
f_sqrt4 <- function(x=NULL){
  if(is.null(x)) x <- 9
  #calculates the squre root of a number
  sqrt(x)
}

#definicija pete funkcije
f_sqrt5 <- function(x){
  if(missing("x")) x <- 9
  #calculates the squre root of a number
  sqrt(x)
}

#definicija seste funkcije
f_sqrt6 <- function(x=9){
  #calculates the squre root of a number
  sqrt(x)
}
```

```r
f_sqrt4()
f_sqrt4(25)
f_sqrt5()
f_sqrt5(25)
f_sqrt6()
f_sqrt6(25)
```

Katera funkcija se vam zdi bolj berljiva? (lazje interpretacije)

Vrnitev vec rezultatov

Napisite funkcijo, ki vrne poleg korena tudi izvirno vrednost (x in sqrt(x)) in vase ime (kot v naslednjem primeru). Namig: funkcije lahko vrnejo samo en "predmet" (object). Ce zelite, da funkcija vrne vec rezultatov, morajo biti rezultati vkljuceni v seznamu (`list`).

```
f_sqrt7(25)
```

```
## $x
## [1] 25
##
## $`Koren x-a`
## [1] 5
##
## $Ime
## [1] "Lara"
```

Definicija razredov (S3) in specificnih metod

Definirajte nov razred (class) za rezultat funkcije `f_sqrt7` (poimenujte ga `listKoren`)

```
x <- f_sqrt7(25)
class(x) <- "listKoren"
```

Ce zelite bolj splosno definicjo razreda znotraj funkcije, lahko uporabite funkcijo `structure(list(my.list), class = "listKoren")` znotraj funkcije `f_sqrt7()`, ali class(my.list) <- "listKoren". (Najprej boste shranili-boste seznam rezultatov v my.list).

Z ukazom print(x), boste dobili spodnji rezultat, ki ni zelo berljv.

```
## $x
## [1] 25
##
## $`Koren x-a`
## [1] 5
##
## $Ime
## [1] "Lara"
##
## attr(,"class")
## [1] "listKoren"
```

Definirajte funkcijo print, ki bo specificna za razred listKoren in ki bo vrnila spodnji rezulta t z ukazom `print(x)`

```
## Stevilka: 25
## Koren: 5
## Avtor funkcije: Lara
```

Oglejte si vse obstojece metode za print z ukazom `methods(print)`. Izberite eno metodo oznaceno z `*` in drugo, ki nima `*` (na primer: `print.anova*` in `print.factor`).
Oglejte si kodo obeh metod. Kaj je razlika? Namig: Naleteli boste na napis: `Non-visible functions are asterisked`, za ogled kode teh metod, boste morali uporabiti funkcijo `getAnywhere()` ali, ce poznate paket, v katerem je funkcija vkljucena, boste lahko uporabili `:::` (na primer: `stats:::print.anova`).

Debugging

Ko napisete funkcijo v R-u, bo v prvih verzijah funkcije skoraj gotovo prisotna kaksna napaka. V R-u lahko poiscete napake v kodi na vec nacinov.

- rocno: natisnite z `print()` ali `cat()` trenutno stanje funkcije na ekranu. (lahko pomaga, ce morate dolociti pri kateri iteraciji se je simulacija ustavila)
- `traceback()`: lahko uporabite ta vkaz po napaki, uporabno za dolocanje, kje se je pojavila napaka (ne zakaj!). R-Studio avtomaticno nudi moznost za traceback (s klikom), ko pride do napake. V R-Studio dobite nekaj vec informacij, ce ste uporabili `source` in je bila funkcija shranjena v datoteki.
- `browser()`: ce napisete ta ukaz znotraj kode, se bo izvedba funkcije ustavila pri tej tocki in si boste lahko ogledali, kaj funkcija vsebuje v nekem danem trenutku. Debugging tako postane interaktivni. Uporabni ukazi: `ls` (poda seznam spremenljivk), `n` (za next, izvede naslednji ukaz), `s` (za step, izvede naslednjo funkcijo), `f` (za finish, gre ven iz funkcije/zanke), `c` (za continue, gre ven iz interaktivnega debuggerja, nadaljuje izvedbo funkcije), `Q` (za quit, zakljuci - ne dokoncuje izvedbe funkcije).

V R-Studiu so ti ukazi dosegljivi tudi preko interakvini meniji, ce uporabite Meni Debug.

- `recover()` ali `options(error=recover)`: napisite ta ukaz pred izvedbo kode, ko se bo naslednjic pojavila napaka, boste ste znasli znotraj funkcije - interaktivni debugging. Izberete `frame` (okolje), ki ga zelite analizirati. Z ukazom `ls` (brez oklepajev) lahko si ogledate, katere spremenljivke so v framu. Napisite njihovo ime in se bo prikazala njihova vsebina. Ustavite debugging z ukazom `Q` ali `0`.

Vrnete se v normalnih nastavitvah z `options(error=stop)` (ko se pojavi napaka, se koda ustavi).

- `debug(ime.funkcije)` napisite ta ukaz pred izvedbo kode, ko se bo naslednjic uporabila funkcija, boste ste znasli znotraj funkcije (od zacekta izvedbe). Uporabili boste iste ukaze kot pri funkciji `browser()`. (Ce ne zelite vec, da se izvede debugging za izbrano funkcijo, morate ga izkljuciti s funkcijo `undebug(ime.funkcije)` ali na novo naloziti funkcijo). Ce zelite uporabiti ta ukaz samo enkrat lahko uporabite funkcijo `debugonce(ime.funkcije)`. Ta opcija je zelo uporabna, ko se napaka pojavi pri vsaki izvedbi funkcije, manj uporabna, ko so napake odvisne od podatkov, ki jih analizirate, na primer samo v nekaterih iteracij simulacij.
- dump.frames(): shrani v zunanjo datoteko (.rda, v mapi, kjer delate) vsebino funkcije, kjer pride do napak. Lahko si ogledate interkativno vsebino s funkcijo `debugger()`. Lahko izvedete debugging kadarkoli, ker vse potrebne kolicine so shranjene.

Primer uporabe:

```
options(error = dump.frames("testdump", TRUE))
#call a function f() that returns an error
f()
load("testdump.rda")
debugger(testdump)
```

Naloga 1

Spodaj je podana funkcija, ki nakljucno izbere stolpce matrike (stevilo izbranih stopcev je lahko od 1 do stevilo stolpcev, stevilo izbranih stolpcev je nakljucno) in izracuna povprecno vrednost vrstic tako definirare matrike.

```
f.rowmeans <- function(my.mat){
  sampled.cols <- unique(sample(1:ncol(my.mat), replace=TRUE))
  apply(my.mat[, sampled.cols], 1, mean)
}


my.matrix <- matrix(1:100, ncol=5)


set.seed(1234) #shranimo zacetno slucajno stevilko za ponovljivost
```

```
#preizkusimo, da funkcija deluje
f.rowmeans(my.matrix)
```

```
##  [1] 47.66667 48.66667 49.66667 50.66667 51.66667 52.66667 53.66667
##  [8] 54.66667 55.66667 56.66667 57.66667 58.66667 59.66667 60.66667
## [15] 61.66667 62.66667 63.66667 64.66667 65.66667 66.66667
```

```
# koda za uporabo z lapply()

#f.rowmeans <- function(index, my.mat){
#  sampled.cols <- unique(sample(1:ncol(my.mat), replace=TRUE))
#  apply(my.mat[, sampled.cols], 1, mean)
#  }


#lapply(1:10000, f.rowmeans, my.matrix)
```

Ocenite povprecje vrstic 1000 krat (spodaj je podana koda s `for` zanko). Pojavila se bo napaka (lahko si ogledate informacije glede napake z opcijami, ki so na voljo v R-Studio (show traceback) ali s funkcijo `traceback()` - preberite najprej spodnja sporocila (manjse stevilke))

```
set.seed(1234) #shranimo zacetno slucajno stevilko za ponovljivost
B <- 1000 #stevilo ponovitev
my.res <- vector("list", B) #seznam, kjer bodo shranjeni rezultati

for(b in 1:B) {
  my.res[[b]] <- f.rowmeans(my.matrix)
}
```

Najdite napako s pomocjo debugginga in jo popravite!

- Pomagajte si z `traceback()`. Ali vam je ze jasno, zakaj pride do napak?

Postopek za debugging v tem primeru

- Z `options(error=recover)` odkrite pri kateri iteraciji je prislo prvic do napake! (Pazite, da boste uporabili zmeraj isti `seed`)
- Ze s tem ukazom lahko odgovorite na vprasanja: kako izgleda matrika z izbranimi stolci pri tej iteraciji? Katere stolpce ste izbrali?
- Primerno dodajte ukaz `browser()` v kodo, da bi si lahko ogledali, kako zgleda matrika in katere stolpce ste izbrali v prvem problematicnem primeru. (se vedno uporabljajte isti seed!)
- Popravite kodo - naredite cim manj sprememb! Namig: oglejte si primere s prvega predavanja, kjer smo omenjali, kako se izogniti tovstnih tezav, ko delamo z matrikami.

Naloga 2

Izvedite gornjo funkcijo z napako, izognite se koncanja izracunanja s funkcijo `try(, silent=TRUE)`

```
my.matrix <- matrix(1:100, ncol=5)
set.seed(1234) #shranimo zacetno slucajno stevilko za ponovljivost


B <- 1000 #stevilo ponovitev
my.res <- vector("list", B) #seznam, kjer bodo shranjeni rezultati
```

```
for(b in 1:B) {
  my.res[[b]] <- try(f.rowmeans(my.matrix), silent=TRUE)
}
```

- Kaj se dogaja pri iteracijah, kjer pride do napak? (Oglejte si, kaj je shranjeno v `my.res`)
- V koliksnih simulacijah se je pojavila napaka? Navedite indekse, kjer pride do napak.
- Izracunajte skupno povprecno vrednost rezultatov. Odstranite iteracije, kjer je prislo do napak.

Naloga 3

- Ce bi zeleli ujeti ne samo napak (`error`), ampak tudi `warning` ali `message`, bi lahko uporabili funkcijo `tryCatch()`. Primer, kjer je funkcija zelo uporabna: ocenimo statisticne modele in pri nekaterih ne pride do konvergence. To se zgodi pogosto pri logisticni regresiji, ko imamo malo dogodkov oz. majhne vzorce. Ceprav dobimo ocenjene modele, rezultati so lahko neveljavni (zelo visoke standardne napake ali ocenjene koeficiente pri logisticni regresiji).

Poglejte, kaj se dogaja v naslednjem primeru.

```
x = c(1:10)
y = c(1,1,0,0,0,0,0,0,0,0)
fit = glm( y~x , family = binomial( link = "logit" ) )
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
summary(fit)
```

```
##
## Call:
## glm(formula = y ~ x, family = binomial(link = "logit"))
##
## Deviance Residuals:
##        Min          1Q      Median          3Q         Max
## -2.694e-05  -2.110e-08  -2.110e-08  -2.110e-08   2.476e-05
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)    109.19  121882.04   0.001    0.999
## x              -43.64   47051.09  -0.001    0.999
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1.0008e+01  on 9  degrees of freedom
## Residual deviance: 1.3389e-09  on 8  degrees of freedom
## AIC: 4
##
## Number of Fisher Scoring iterations: 25
```

V takih primerih lahko dolocimo, ce ni prislo do konvergence, oziroma ce je nastala kaksna druga tezava, s pomocjo funkcije tryCatch. To je posebej pomembno v simulacijah, kjer ocenimo ogromno modelov in bi zeleli vedeti pri katerih ni prislo do konvergence.

```
res.fit <- tryCatch(  { fit = glm( y~x , family = binomial( link = "logit" ) ) } , warning = function(e
```

```
## [1] "Warning in fit"
```

Generirajte 1000 datotek velikosti n=10: pojasnjevalna spremenljivka naj bo generirana iz N(0,1), dihotmni izid iz binomske porazdelitve z verjetnostjo izida 0.10, izid in pojasnjevalna spremenljivka nista povezani (veljavna nicelna domneva). Za ponovljivost uporabite ukaz `set.seed(1234)`.

Za vsako datoteko ocenite model logisticne regresije (`fit <- glm(y~x,family=binomial(link="logit"))`) in shranite ocenjene regresijske (`fit$coef`) koeficiente.

Hkrati shranite tudi pri katerih iteracijah je program javil warning - ni prislo do konvergence, ocenjene verjetnosti so natanko 0 in/ali 1. Porocajte steviko takih primerov. Primerjajte povprecne ocenejene koeficiente za vse modele in loceno za tiste, kjer je R javil warning-e. Kaj opazite?

```
## [1] 0.096
```

```
##        V1                 V2
##  Min.   :-24.5661   Min.   :-15.1312
##  1st Qu.:-24.5661   1st Qu.: -0.2555
##  Median : -2.5858   Median :  0.0000
##  Mean   :-10.3662   Mean   :  0.0195
##  3rd Qu.: -1.7675   3rd Qu.:  0.2513
##  Max.   :  0.1884   Max.   :  9.1332
```

```
##        V1                  V2
##  Min.   :-13745.78   Min.   :-11956.26
##  1st Qu.:  -244.92   1st Qu.:  -102.02
##  Median :  -129.76   Median :    30.89
##  Mean   :  -468.57   Mean   :   -58.55
##  3rd Qu.:   -74.37   3rd Qu.:   117.41
##  Max.   :    10.40   Max.   : 10354.98
```

```
##        V1                 V2
##  Min.   :-24.5661   Min.   :-15.1312
##  1st Qu.:-24.5661   1st Qu.: -0.2555
##  Median : -2.5858   Median :  0.0000
##  Mean   :-10.3662   Mean   :  0.0195
##  3rd Qu.: -1.7675   3rd Qu.:  0.2513
##  Max.   :  0.1884   Max.   :  9.1332
```

```
##
##                       glm.fit: algorithm did not converge
##                                                        66
## glm.fit: fitted probabilities numerically 0 or 1 occurred
##                                                        30
```

Naloga 4

Oglejte si spodnji primer uporabe funkcije `predict` pri logisticni regresiji. Podatke generiramo za 100 statisticnih enot (training) in s temi ocenimo model, ki ga zelimo uporabiti na 500 novih podatkov (test). Tokrat je binarni izid povezan s 5 pojasnjevalnimi spremenljivkami, v model dodamo se 5 nicelnih pojasnjevalnih spremenljivk (verjetnost izida bo odvisen od vrednosti ne-nicelnih pojasnjevalnih spremenljivk).

```
b1<-1 #value of the regression coefficient for the covariates with non-null effect
numdif <- 5 #number of covariates with non-null effect
p <- 10 #overall number of covarites

n <- 100
ntest <- 500
```

```
 a <- 0 #intercept term

 beta<-matrix(c(a,rep(b1,numdif),rep(0,p-numdif)),ncol=1)

  #training

  x<-cbind(rep(1,n), matrix(rnorm(n*p),ncol=p)) #covariates

  x <- as.matrix(x)
 lp<-x%*%beta #linear predictor on training data
 ptr<-exp(lp)/(1+exp(lp)) #estimated probabilities on training data


  y<-as.factor(rbinom(n,size=1,prob=ptr)) #outcome data, depends on the probability of event

  y1<-as.numeric(y)-1
  if(any(!is.finite(y))) y[!is.finite(y)]<- ifelse(ptr[!is.finite(y)]<mean(y1), 0, 1)
  y1<-as.numeric(y)-1

  X <- x[,-1] #removing the intercept
  fit<-glm(y~x, family="binomial")

  #fit<-glm(y~., family="binomial", data=as.data.frame(x)) #logistic regression model

  xte<-cbind(rep(1,n),matrix(rnorm(ntest*p),ncol=p)) #test data for the covariates

  lpte<-xte%*%beta #linear predictor for the test data
  pte<-exp(lpte)/(1+exp(lpte)) #event probabilities for the test data

  yte <- rbinom(ntest,size=1,prob=pte) #outcome data from the test data



################### prediction does not work properly!!!!!!!!!!!!!!!!!!!!!!!!!!!!
  pred<-predict(fit, newdata=as.data.frame(xte),type="response")
```

```
## Warning: 'newdata' had 500 rows but variables found have 100 rows
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type =
## ifelse(type == : prediction from a rank-deficient fit may be misleading
```

Kaj je narobe? Ali so napovedane vrednosti pravilne? Ali bi se zavedali modebitnih tezav, ce bi bila velikost testne skupine ista kot velikost podatkov, ki ste jih uporabili za oceno modela?

Popravite gornjo kodo tako, da se bodo napovedane vrednosti nanasale na testne podatke!