

Funkcije in debugging (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 square root of a number
  sqrt(x)
}

body(f_sqrt) #komentarji niso vključeni

## {
##   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 square 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 square root of a number
  sqrt(x)
}

#definicija tretje funkcije
```

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

```
x <- 25
```

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

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

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

Zelo pogosto ima uporabnik x v splošnem okolju in pozabi definirati kaksen od “notranjih” parametrov. R najde x v splošnem okolju, ne javi napake, ki se bo pojavila, ko bo novi uporabnik želel 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`.

```
library(codetools)
codetools::findGlobals(f_sqrt)
codetools::findGlobals(f_sqrt2)
codetools::findGlobals(f_sqrt3)
```

Napovejte rezultat - uporaba predefiniranih vrednosti oz. neobveznih parametrov

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

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

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

```
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

Napišite 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). Če želite, da funkcija vrne več rezultatov, morajo biti rezultati vključeni v seznamu (list).

```
f_sqrt7 <- function(x){  
  #calculates the square root of a number  
  return(list(x=x, `Koren x-a`=sqrt(x), Ime="Lara"))  
}
```

```
f_sqrt7(25)
```

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

Definicija razredov (S3) in specifičnih metod

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

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

Če želite bolj splošno definicijo 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 berljiv.

```
print(x)
```

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

Definirajte funkcijo `print`, ki bo specifična za razred `listKoren` in ki bo vrnila spodnji rezultat z ukazom `print(x)`

```
print.listKoren <- function(l){  
  #calculates the square root of a number  
  cat("Stevilka:", l[[1]], "\n")  
}
```

```
cat("Koren:", 1[[2]], "\n")
cat("Avtor funkcije:", 1[[3]], "\n")
}
```

```
print(x)
```

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

Oglejte si vse obstoječe metode za print z ukazom `methods(print)`. Izberite eno metodo označeno 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, če poznate paket, v katerem je funkcija vključena, boste lahko uporabili `:::` (na primer: `stats:::print.anova`).

```
print.factor
getAnywhere(print.anova)
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.

- **ročno:** natisnite z `print()` ali `cat()` trenutno stanje funkcije na ekranu. (lahko pomaga, če morate določiti pri kateri iteraciji se je simulacija ustavila)
- **traceback():** lahko uporabite ta vkaž po napaki, uporabno za določanje, kje se je pojavila napaka (ne zakaj!). R-Studio avtomatično nudi možnost za traceback (s klikom), ko pride do napake. V R-Studio dobite nekaj več informacij, če ste uporabili `source` in je bila funkcija shranjena v datoteki.
- **browser():** če napisete ta ukaz znotraj kode, se bo izvedba funkcije ustavila pri tej točki 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, zaključí - ne dokončuje izvedbe funkcije).

V R-Studio so ti ukazi dosegljivi tudi preko interaktivni meniji, če 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()**. (Če ne zelite vec, da se izvede debugging za izbrano funkcijo, morate ga izkljuciti s funkcijo **undebug(ime.funkcije)** ali na novo naloziti funkcijo). Če 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 interaktivno 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 naključno izbere stolpce matrike (stevilo izbranih stolpcev je lahko od 1 do stevilo stolpcev, stevilo izbranih stolpcev je naključno) in izračuna povprečno vrednost vrstic tako definirane 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 povprečje 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 sporočila (manjše številke))

```
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 pomočjo debuginga in jo popravite!

- Pomagajte si z `traceback()`. Ali vam je že 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 vprašanja: 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 tovrstnih težav, ko delamo z matrikami.

```
options(error=recover)
#izberite frame 4
#ukaz: ls()
#b: bo podal iteracijo pri kateri ste se ustavili
#b je 293

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) {
  if(b==293) browser()
  my.res[[b]] <- f.rowmeans(my.matrix)
}

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

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.ok(my.matrix)
}
```

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`)

```
my.res[[293]]
class(my.res[[293]])
```

- V koliksnih simulacijah se je pojavila napaka? Navedite indekse, kjer pride do napak.

```
library(magrittr)
class.my.res <- my.res %>%
  lapply(class) %>%
  unlist()

#equivalent, sapply does the unlisting

class.my.res <- my.res %>%
  sapply(class)

which.indexes <- which(class.my.res=="try-error")

#equivalent, without using the pipe operator from the magrittr package
class.my.res <- sapply(my.res, class)
which.indexes <- which(class.my.res=="try-error")

#equivalent, with one line
which.indexes <- which(sapply(my.res, function(x) class(x)=="try-error"))
```

- Izračunajte skupno povprečno vrednost rezultatov. Odstranite iteracije, kjer je prislo do napak.

```
my.mean <- my.res[-which.indexes] %>%
  unlist() %>%
  mean()
```

Naloga 3

- Če bi zeleli ujeti ne samo napak (**error**), ampak tudi **warning** ali **message**, bi lahko uporabili funkcijo **tryCatch()**. Primer, kjer je funkcija zelo uporabna: ocenimo statistične modele in pri nekaterih ne pride do konvergence. To se zgodi pogosto pri logistični regresiji, ko imamo malo dogodkov oz. majhne vzorce. Čeprav dobimo ocenjene modele, rezultati so lahko neveljavni (zelo visoke standardne napake ali ocenjene koeficiente pri logistični 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 določimo, če ni prislo do konvergence, oziroma če je nastala kaksna druga težava, s pomočjo 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) { print("Warning in fit") } )
```

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

Generirajte 1000 datotek velikosti $n=10$: pojasnjevalna spremenljivka naj bo generirana iz $N(0,1)$, dihotomni izid iz binomske porazdelitve z verjetnostjo izida 0.10, izid in pojasnjevalna spremenljivka nista povezani (veljavna ničelna 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 povprečne ocenjene koeficiente za vse modele in loceno za tiste, kjer je R javil warning-e. Kaj opazite?

```
B <- 1000 #number of iterations
#saving the warnings
w<-numeric(B)
w2 <- vector("list", B)

my.coef <- matrix(NA, ncol=2, nrow=B) #matrix where the regression coefficients will be stored

set.seed(1234)
n <- 10 #velikost vzorca

for(b in 1:B){

  #generiramo podatke
  y <- rbinom(n, prob=0.10, size=1)
  x <- rnorm(n)

  fit<-glm(y~x,family=binomial(link="logit"))

  my.coef[b,] <- fit$coefficients
```



```

my.warn<- tryCatch( {my.mod <- glm(y~x,family=binomial(link="logit"))},
  warning=function(war) {
    w[b]<=-1
    w2[[b]]<=-war
  }
)
}

```

```

#delez iteracij z warningom
mean(w)

```

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

```

#usi modeli
summary(my.coef[w==0,])

```

```

##           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

```

```

#modeli s tezavami glede konvergence
summary(my.coef[w==1,])

```

```

##           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

```

```

#modeli, kjer ni bilo warningov
summary(my.coef[w==0,])

```

```

##           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

```

```

#razlog za warning
table(unlist(w2[w==1]))

```

```

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

```

```
#Kaj bi lahko storili: uporabili Firthov popravek za oceno logisticne regresije, library(logistf)
#if(w[ii]==1) fit<-logistf(y~x)
```

Naloga 4

Oglejte si spodnji primer uporabe funkcije `predict` pri logisticni regresiji. Podatke generiramo za 100 statističnih enot (training) in s temi ocenimo model, ki ga želimo 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!

```
#### one possible solution
```

```
X<-(x[, -1])
X<-data.frame(X)
# dimnames(X)[[2]] <- paste0("X", 1:p)

fit<-glm(y~., family="binomial", data=X)

coefsi<-coef(fit)

XTE<-(xte[, -1])
XTE <-data.frame(XTE)
dimnames(XTE)[[2]] <- dimnames(X)[[2]] #paste0("X", 1:p)
#paste("X", 1:p, sep=".")

pred<-predict(fit, newdata=XTE, type="response")
```