Modelos de marcação e recaptura: populações fechadas

Contents

- Arquivo em pdf
- Arquivo em markdown (para executar os comandos no R studio)

Preparação

Vamos usar o pacote *RMark*, que é um pacote do R para usar o programa MARK. Siga as instruções deste sítio para instalar o MARK para uso pelo pacote: (http://www.phidot.org/software/mark/rmark/).

Abra o R e carregue o pacote

```
library(RMark)
```

Usaremos dados de registro fotográfico de indivíduos do boto cinza (*Sotalia guianensis*) em 11 ocasiões. Aqui há mais informações sobre este caso de estudo.

Os dados estão no formato nativo do MARK (*.inp*). Use os comandos abaixo para importá-lo para o R:

```
## Link dos dados na página da disciplina
url <- "http://ecologia.ib.usp.br/bie5703/lib/exe/fetch.php?media=roteiros:botos_2002.inp"
## Importa arquivo inp
boto2002 <- convert.inp(url)
```

Ajuste dos modelos

Processamento dos dados

O primeiro passo é usar a função `process.data` para criar um objeto com as informações que o Mark usa para ajustar o modelo. Uma delas é o tipo de modelo, que é indicado no argumento `model`.

Para o modelo de populações fechada sem heterogeneidade e de verossimilhança não condicionada este argumento é `model="Closed"`:

```
boto <- process.data(data=boto2002, model="Closed")
```

E para o modelo com heterogeneidade o argumento é `model="FullHet"`

```
botoH <- process.data(data=boto2002, model="FullHet")
```

Ajuste dos modelos sem heterogeneidade

Para ajustar os modelos, crie listas que especificam a fórmula de cada termo. No modelo `Closed` os nomes parâmetros que podem variar são `p` (probabilidade da primeira captura), `c` (probabilidade de recaptura). O objeto criado na seção acima tem uma covariável de tempo chamada `time`, que então pode ser usado nas fórmulas:

```
## Fórmulas estatísticas para cada parâmetro do modelo sem heterogeneidade
## p e c constantes mas diferentes
t.dot <- list(formula=~1)
## p=c contantes (use o argumento share=TRUE)
t.dotshared=list(formula=~1,share=TRUE)
## Parametros dependem do tempo
t.time <- list(formula=~time)
## Parametro p=c dependem do tempo
t.timeshared <- list(formula=~time, share=TRUE)
```

E usamos a função `mark` para fazer os ajuste:

```
boto.M0 <- mark(boto, model.parameters=list(p=t.dotshared))
boto.Mb <- mark(boto, model.parameters=list(c=t.dot, p=t.dot))
```

```
##
## Note: only 2 parameters counted of 3 specified parameters
##
## AICc and parameter count have been adjusted upward
```

```
boto.Mt <- mark(boto, model.parameters=list(p=t.timeshared))
boto.Mtb <- mark(boto, model.parameters=list(c=t.time, p=t.time))
```

```
##
## Note: only 18 parameters counted of 22 specified parameters
##
## AICc and parameter count have been adjusted upward
```

Ajuste dos modelos com heterogeneidade

Para os modelos com heterogeneidade acrescente o termo `mixture` nas fórmulas do parâmetro p:

```
## Fórmulas estatísticas para cada parâmetro do modelo com heterogeneidade
## p com heterogeneidade
t.mix <- list(formula=~mixture)
## p=c com heterogeneidade (use o argumento share=TRUE)
t.mixshared=list(formula=~mixture,share=TRUE)
## Parametros dependem do tempo
t.timemixshared <- list(formula=~time+mixture, share=TRUE)
t.timemix <- list(formula=~time+mixture)
```

E ajuste os modelos

```
boto.Mh <- mark(botoH, model.parameters=list(p=t.mixshared))
boto.Mbh <- mark(botoH, model.parameters=list(c=t.mix, p=t.mix))
boto.Mth <- mark(botoH, model.parameters=list(p=t.timeshared))
boto.Mtbh <- mark(botoH, model.parameters=list(c=t.timemix, p=t.timemix))
```

```
##
## Note: only 22 parameters counted of 25 specified parameters
##
## AICc and parameter count have been adjusted upward
```

Seleção de modelos

A função abaixo retorna a tabela de seleção de modelos:

```
collect.models(lx=c("boto.M0", "boto.Mb", "boto.Mt", "boto.Mtb",
                    "boto.Mh", "boto.Mbh", "boto.Mth", "boto.Mtbh"))
```

```
## Warning in model.table(x, type, pf = 2, adjust = adjust): Model list contains models of differing ty
```

```
##                                              model npar     AICc DeltaAICc
## 8 pi(~1)p(~time + mixture)c(~time + mixture)f0(~1)   25 263.7916  0.000000
## 3                                 p(~time)c()f0(~1)   12 270.1989  6.407315
## 7                           pi(~1)p(~time)c()f0(~1)   12 270.1989  6.407315
## 6               pi(~1)p(~mixture)c(~mixture)f0(~1)    6 275.2837 11.492157
## 5                       pi(~1)p(~mixture)c()f0(~1)    4 275.6888 11.897169
## 4                           p(~time)c(~time)f0(~1)   22 282.6256 18.833993
## 1                                   p(~1)c()f0(~1)    2 289.7349 25.943289
## 2                                 p(~1)c(~1)f0(~1)    3 290.1108 26.319260
##         weight Deviance
## 8 9.198640e-01 156.7423
## 3 3.735880e-02 191.7698
## 7 3.735880e-02 191.7698
## 6 2.939230e-03 209.4365
## 5 2.400414e-03 213.9520
## 4 7.481247e-05 182.3529
## 1 2.138996e-06 232.0679
## 2 1.772430e-06 230.4140
```

Valores das estimativas

A função `coef` retorna os coeficientes na escala de ligação (logito). Para as estimativas na escala de probabilidades use a função `get.real`:

```
coef(boto.Mtbh, data=boto2002)
```

```
##                    estimate        se         lcl         ucl
## pi:(Intercept)  -1.8513506 0.5803075  -2.9887534  -0.7139478
## p:(Intercept)    1.8247866 0.0725632   1.6825627   1.9670104
```

```
## p:time2          0.3147905 0.4532426  -0.5735650   1.2031460
## p:time3          0.5439946 0.5149130  -0.4652349   1.5532240
## p:time4          0.9695441 0.6113591  -0.2287198   2.1678079
## p:time5         -0.4569549 1.0985804  -2.6101725   1.6962627
## p:time6          0.7470255 0.9166179  -1.0495456   2.5435967
## p:time7          0.4593326 1.2275442  -1.9466541   2.8653193
## p:time8        -20.8650940 0.0000000 -20.8650940 -20.8650940
## p:time9        -20.8650940 0.0000000 -20.8650940 -20.8650940
## p:time10         1.1524707 1.4166330  -1.6241302   3.9290715
## p:time11        37.8892030 0.9732125  35.9817060  39.7966990
## p:mixture2      -2.9772937 0.1013092  -3.1758597  -2.7787277
## c:(Intercept)    0.6699954 0.8294849  -0.9557950   2.2957858
## c:time3          0.1286742 0.9168714  -1.6683938   1.9257422
## c:time4          1.3448816 0.8531357  -0.3272643   3.0170275
## c:time5          1.4845221 0.8366733  -0.1553575   3.1244017
## c:time6          0.3161753 0.8530056  -1.3557157   1.9880663
## c:time7          1.0357332 0.8315229  -0.5940517   2.6655181
## c:time8         -1.8531905 1.0575826  -3.9260524   0.2196714
## c:time9         -0.6810178 0.8999993  -2.4450166   1.0829809
## c:time10        -0.9872526 0.9283738  -2.8068653   0.8323602
## c:time11        -0.7041478 0.8988334  -2.4658613   1.0575656
## c:mixture2      -2.3847890 0.4622453  -3.2907899  -1.4787881
## f0:(Intercept) -18.2865930 0.0000000 -18.2865930 -18.2865930
```

```
## Na escala de probabilidades
get.real(boto.Mtbh, parameter="p")
```

```
## [[1]]
##                   1         2         3         4         5         6
## mixture:1 0.8611395 0.8946908 0.9144155 0.9423687 0.7970296 0.9290253
## mixture:2 0.2400314 0.3020159 0.3523986 0.4543864 0.1666633 0.3999960
##                   7            8            9        10 11
## mixture:1 0.9075532 5.381452e-09 5.381452e-09 0.9515360  1
## mixture:2 0.3333272 2.740799e-10 2.740799e-10 0.4999909  1
```

```
get.real(boto.Mtbh, parameter="c")
```

```
## [[1]]
##                   2         3         4         5         6         7
## mixture:1 0.6615021 0.6896898 0.8823502 0.8960902 0.7283309 0.8462814
## mixture:2 0.1525430 0.1699306 0.4085623 0.4426852 0.1980355 0.3364710
##                   8         9        10        11
## mixture:1 0.2344782 0.4972444 0.4213443 0.4914627
## mixture:2 0.0274386 0.0834927 0.0628527 0.0817397
```

```
get.real(boto.Mtbh, parameter="pi")
```

```
## [[1]]
##
## mixture:1 0.1357144
```