

Manual de detección automática de vocalizaciones de ratas en R

Proyecto C0754 Desarrollo de una herramienta computacional para la detección, la cuantificación estructural y la clasificación semisupervisada de vocalizaciones ultrasónicas de ratas

Centro de Investigación en Neurociencias, Universidad de Costa Rica

Marcelo Araya Salas, Juan Carlos Brenes Saenz & Mijail Rojas Carvajal

15-11-2021

Contenidos

Preparar los archivos de audio	2
Detección automática de llamados ultrasónicos	3
Vocalizaciones de 55 kHz en prueba de campo abierto (sin borucha)	3
Vocalizaciones de 55 kHz en prueba de jaula (con borucha)	5
Vocalizaciones de 22 kHz en prueba de campo abierto (sin borucha)	5

Este manual describe los pasos necesarios para la detección automática de vocalizaciones de ratas. La detección se lleva a cabo usando como herramienta principal el paquete de R [ohun](#), el cual facilita la detección automática de señales acústicas proporcionando funciones para diagnosticar y optimizar las rutinas de detección.

Primero debemos instalar y cargar los paquetes [ohun](#) y [ranger](#):

```
# instalar
devtools::install_github("maRce10/ohun")
install.packages("ranger")

# cargar
library(ohun)
library(ranger)
```

Preparar los archivos de audio

Siempre es importante asegurarse que los archivos de audio pueden ser leídos en R. Esto lo podemos hacer así:

```
# definir ruta
ruta_archivos <- "RUTA DONDE SE ENCUENTRAN LOS ARCHIVOS"

# revisar archivos
check_wavs(path = ruta_archivos)

## All files can be read
```

Si todo está bien, el mensaje devuelve el mensaje “All files can be read”. Note que el argumento `path` debe ser usado para indicar el directorio que contiene los archivos de audio. Este argumento sera usado por la mayoría de las funciones y es buena idea definir un valor desde el inicio.

Las rutinas de detección que se detallan en este manual pueden tomar un tiempo considerable en aplicarse (e.g. > 1 hora). Esto debido a que las grabaciones de sonidos ultrasónicos tienen tasas de muestreo muy altas, lo que hace que los archivos sean muy pesados. Una forma de mejorar la velocidad de la detección es reducir la tasa de muestreo de los archivos. Para las vocalizaciones de ratas una tasa de muestreo de 200 kHz es suficiente para que las llamadas sean registradas con precisión. Podemos bajar la tasa de muestreo de esta forma:

```
# cambiar tasa de muestreo
fix_wavs(samp.rate = 200, path = ruta_archivos)
```

Si desconocemos la tasa de muestreo actual de nuestros archivos podemos revisarla de esta forma:

```
# ver información de archivos
wav_info(path = ruta_archivos)
```

##	sound.files	duration	sample.rate	channels	bits	wav.size	samples
## 1	T00000001-1.wav	300.0000	200	1	16	234.3750	60000000
## 2	T00000001-10.wav	300.0000	200	1	16	234.3750	60000000
## 3	T00000001-11.wav	300.0000	200	1	16	234.3750	60000000
## 4	T00000001-12.wav	160.9562	200	1	16	125.7471	32191232

Los archivos de audio de larga duración (20 min o mas) pueden generar problemas durante la detección (noten que este atributo también se puede revisar con `wav_info()`). En estos casos es recomendable segmentarlos en archivos de menor duración. Esto lo podemos hacer de la siguiente forma:

```
# dividir en segmentos de 5 min
metadatos_nuevos_archivos <- split_sound_files(sgmt.dur = 5 *
  60, path = ruta_archivos)
```

El código anterior divide todos los archivos de audio en archivos de 5 minutos. El objeto `metadatos_nuevos_archivos` que se produjo contiene los nombres de los nuevos archivos así como de cual archivo provienen.

Detección automática de llamados ultrasónicos

La detección automática se lleva a cabo por medio de dos procesos:

1. Detección de sonidos con umbrales de energía: para una descripción detallada de este método ver el [caso de estudio del paquete ohu](#). La detección se lleva a cabo con la función `energy_detector()`.
2. Filtrado de los sonidos detectados con *Random Forest*: este paso toma las detecciones producidas en el primer paso junto con medidas de su estructura acústica (e.g. duración, frecuencia, distribución de energía) y genera un modelo que distingue las llamadas ultrasónicas del ruido de fondo. En otras palabras filtra las señales de interés de otros sonidos no deseados.

La actividad de investigación en la que se desarrollaron las rutinas de detección que se muestran a continuación, también permitió identificar los parámetros de ajuste (“tuning parameters”) que optimizan la detección para diferentes contextos experimentales (prueba de jaula y campo abierto) y categorías de vocalizaciones (22 kHz y 55 kHz). Este manual se limita a presentar las funciones utilizadas para la detección en estos diferentes escenarios junto con los parámetros de ajuste que produjeron el mejor desempeño en cada escenario. Los modelos de *Random Forest* para cada escenario se encuentran en [este repositorio en línea](#).

Vocalizaciones de 55 kHz en prueba de campo abierto (sin borucha)

Detección con umbrales de energía:

```
# detectar
deteccion <- energy_detector(threshold = 1, min.duration = 1,
  ssmooth = 5, hold.time = 5, thinning = 0.5, parallel = 1,
  bp = c(35, 90), max.duration = 15, path = ruta_archivos)
```

Medición de parámetros acústicos para el *Random Forest*:

```
# medir parámetros acústicos
parametros_acusticos <- spectro_analysis(deteccion,
  bp = c(35, 85), fast = TRUE, ovlp = 70, parallel = 1,
  path = ruta_archivos)
```

Clasificación con *Random Forest*:

```
# bajar modelo de random forest del repositorio
# de figshare
download.file(url = "https://figshare.com/ndownloader/files/31475039",
  destfile = "modelo_random_forest_55_kHz_campo_abierto.RDS")

# leer el modelo
clasificador_rf <- readRDS("modelo_random_forest_55_kHz_campo_abierto.RDS")

# aplicarlo sobre las detecciones nuevas
deteccion$clase <- predict(object = clasificador_rf,
  data = parametros_acusticos)$predictions

# remover los sonidos clasificados como ruido de
# fondo
deteccion_filtrada <- deteccion[deteccion$clase == "true.positive",
  ]
```

El objeto resultante (`deteccion_filtrada`) es un cuadro de datos (“data.frame”) con la posición en el tiempo de las vocalizaciones detectadas para cada uno de los archivos de audio en la ruta proporcionada. Tiene una estructura similar a esta:

sound.files	channel	selec	start	end
Phae.long1.wav	1	1	0.34	0.54
Phae.long1.wav	1	2	1.16	1.34
Phae.long1.wav	1	3	2.15	2.33
Phae.long2.wav	1	4	0.16	0.31
Phae.long2.wav	1	5	0.76	0.88
Phae.long2.wav	1	6	1.46	1.59

Con este cuadro de datos podemos calcular fácilmente parámetros sobre la actividad vocal, como por ejemplo el número de vocalizaciones por archivo:

```
tapply(lbh_selec_reference$sound.files, lbh_selec_reference$sound.files,
  length)

## Phae.long1.wav Phae.long2.wav Phae.long3.wav Phae.long4.wav
##              3              3              4              5
```

Vocalizaciones de 55 kHz en prueba de jaula (con borucha)

Detección con umbrales de energía:

```
# detectar
deteccion <- energy_detector(threshold = 2.5, min.duration = 1,
  ssmooth = 1, hold.time = 3, path = .Options$warbleR$path,
  thinning = 0.5, parallel = 1, bp = c(35, 90), max.duration = 200,
  path = ruta_archivos)
```

Medición de parámetros acústicos para el *Random Forest*:

```
# medir parametros acústicos
parametros_acusticos <- spectro_analysis(deteccion,
  bp = c(35, 85), fast = TRUE, ovlp = 70, parallel = 1,
  path = ruta_archivos)
```

Clasificación con *Random Forest*:

```
# bajar modelo de random forest del repositorio
# de figshare
download.file(url = "https://figshare.com/ndownloader/files/31475096",
  destfile = "modelo_random_forest_55_kHz_prueba_de_jaula.RDS")

# leer el modelo
clasificador_rf <- readRDS("modelo_random_forest_55_kHz_prueba_de_jaula.RDS")

# aplicarlo sobre las detecciones nuevas
deteccion$clase <- predict(object = clasificador_rf,
  data = parametros_acusticos)$predictions

# remover los sonidos clasificados como ruido de
# fondo
deteccion_filtrada <- deteccion[deteccion$clase == "true.positive",
  ]
```

Vocalizaciones de 22 kHz en prueba de campo abierto (sin borucha)

Detección con umbrales de energía:

```
# detectar
deteccion <- energy_detector(files = unique(split_sels$sound.files),
  threshold = 2, min.duration = 2, ssmooth = 17,
```

```
hold.time = 25, path = .Options$warbleR$path, thinning = 0.5,
parallel = 1, bp = c(20, 30), max.duration = 3000,
path = ruta_archivos)
```

Medición de parámetros acústicos para el *Random Forest*:

```
# medir parametros acústicos
parametros_acusticos <- spectro_analysis(deteccion,
  bp = c(20, 30), fast = TRUE, ovlp = 70, parallel = 1,
  path = ruta_archivos)
```

Clasificación con *Random Forest*:

```
# bajar modelo de random forest del repositorio
# de figshare
download.file(url = "https://figshare.com/ndownloader/files/31475099",
  destfile = "modelo_random_forest_55_kHz_prueba_de_jaula.RDS")

# leer el modelo
clasificador_rf <- readRDS("modelo_random_forest_22_kHz_campo_abierto.RDS")

# aplicarlo sobre las detecciones nuevas
deteccion$clase <- predict(object = clasificador_rf,
  data = parametros_acusticos)$predictions

# remover los sonidos clasificados como ruido de
# fondo
deteccion_filtrada <- deteccion[deteccion$clase == "true.positive",
  ]
```

Las rutinas se pueden acelerar usando el argumento **parallel**, el cual se encuentra en varias de las funciones mostradas. Este argumento define el número de núcleos de la computadora (“cores”) que se usarán para correr las diferentes funciones. Valores mayores a uno paralelizan las rutinas (i.e. corren varias tareas a la vez), reduciendo el tiempo de análisis.

Información de la sesión de R

```

## R version 4.1.0 (2021-05-18)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 20.04.2 LTS
##
## Matrix products: default
## BLAS:   /usr/lib/x86_64-linux-gnu/atlas/libblas.so.3.10.3
## LAPACK: /usr/lib/x86_64-linux-gnu/atlas/liblapack.so.3.10.3
##
## locale:
##  [1] LC_CTYPE=pt_BR.UTF-8      LC_NUMERIC=C
##  [3] LC_TIME=es_CR.UTF-8      LC_COLLATE=pt_BR.UTF-8
##  [5] LC_MONETARY=es_CR.UTF-8  LC_MESSAGES=pt_BR.UTF-8
##  [7] LC_PAPER=es_CR.UTF-8     LC_NAME=C
##  [9] LC_ADDRESS=C             LC_TELEPHONE=C
## [11] LC_MEASUREMENT=es_CR.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats      graphics  grDevices  utils      datasets  methods    base
##
## other attached packages:
##  [1] formatR_1.11      kableExtra_1.3.4  ohun_0.1.0        warbleR_1.1.27
##  [5] NatureSounds_1.0.4 knitr_1.33        seewave_2.1.8     tuneR_1.3.3.1
##  [9] devtools_2.4.2    usethis_2.0.1
##
## loaded via a namespace (and not attached):
##  [1] xfun_0.25          remotes_2.4.0      purrr_0.3.4        pbapply_1.4-3
##  [5] colorspace_2.0-2   testthat_3.0.4     viridisLite_0.4.0  htmltools_0.5.2
##  [9] yaml_2.2.1         rlang_0.4.11       pkgbuild_1.2.0     glue_1.4.2
## [13] withr_2.4.2        sessioninfo_1.1.1  lifecycle_1.0.0    stringr_1.4.0
## [17] munsell_0.5.0      rvest_1.0.1        memoise_2.0.0      evaluate_0.14
## [21] callr_3.7.0        fastmap_1.1.0      fftw_1.0-6         ps_1.6.0
## [25] parallel_4.1.0     Rcpp_1.0.7         scales_1.1.1       cachem_1.0.5
## [29] desc_1.3.0         pkgload_1.2.1      webshot_0.5.2      systemfonts_1.0.2
## [33] fs_1.5.0           rjson_0.2.20       digest_0.6.27      stringi_1.7.4
## [37] processx_3.5.2     dtw_1.22-3         rprojroot_2.0.2    cli_3.0.1
## [41] tools_4.1.0        bitops_1.0-7       magrittr_2.0.1     RCurl_1.98-1.4
## [45] proxy_0.4-26       crayon_1.4.1       MASS_7.3-54        ellipsis_0.3.2
## [49] xml2_1.3.2         prettyunits_1.1.1  svglite_2.0.0      rmarkdown_2.10
## [53] httr_1.4.2         rstudioapi_0.13    R6_2.5.0           signal_0.7-7
## [57] compiler_4.1.0

```