Manual de detección automatica de vocalizaciones de ratas en R

Centro de Investigación en Neurociencias

Marcelo Araya-Salas, PhD

15-11-2021

Contenidos

Este manual describe los pasos necesarios para la detección automática de vocalizaciones de ratas. La deteccion 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 el paquete ohun:

```
# instalar
devtools::install_github("maRce10/ohun")

# cargar
library(ohun)
```

Preparar los archivos de audio

Siempre es importante asegurarse que los archivos de audio pueden ser leidos 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 mayoria 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 informacion de archivos
wav_info(path = ruta_archivos)
```

sound.files	duration	sample.rate	channels	bits	wav.size	samples
T0000001-1.wav	300.0000	200	1	16	234.3750	60000000
T0000001-10.wav	300.0000	200	1	16	234.3750	60000000
T0000001-11.wav	300.0000	200	1	16	234.3750	60000000
T0000001-12.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 actividad de investigacion en la que se desarrollaron las rutinas de deteccion que se muestran a continuacion, tambien permitio indetificar los parametros de ajuste ("tuning parameters") que optimizan la deteccion para diferentes contextos experimentales (prueba de jaula y campo abierto) y categorias de vocalizaciones (22 y 55 kHz). Este tutorial se limita a presentar las funciones utilizadas para la deteccion en estos diferentes escenarios junto con los parametros de ajuste que produjeron el mejor desempeño en cada escenario.

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 ohun. La detección se lleva a acabo 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 acustica (e.g. duracion, frequencia, distribucion de energia) y genera un modelo que distingue las llamadas ultrasonicas del ruido de fondo. En otras palabras filtra las señales de interes de otros sonidos no deseados.

Vocalizaciones de 55 kHz campo abierto

Detección con umbrales de energía:

```
# measure spectrographic parameters
parametros_acusticos <- spectro_analysis(filter_ed_all,
    bp = c(35, 85), fast = TRUE, ovlp = 70, parallel = 1)

deteccion <- energy_detector(threshold = 0.01, min.duration = 1,
    ssmooth = 5, hold.time = 5, path = ruta_archivos,
    thinning = 0.5, parallel = 1, bp = c(35, 90), max.duration = 15)
```

Medicion de parametros acusticos para el Random Forest:

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

Clasificación con Random Forest:

```
## leer el modelo del repositorio de github
clasificador_rf <- readRDS("")

# aplicarlo sobre las detecciones nuevas
deteccion$clase <- predict(clasificador_rf, parametros_acusticos,
    type = "terminalNodes")$predictions

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

El objeto resultante (`detecion_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	0.341644	0.5383379
Phae.long1.wav	1	2	1.162889	1.3423291
Phae.long1.wav	1	3	2.153289	2.3258275
Phae.long2.wav	1	4	0.160489	0.3123229
Phae.long2.wav	1	5	0.764400	0.8817262
Phae.long2.wav	1	6	1.458000	1.5856785

Las rutinas se pueden correr usando el argumento `parallel`, el cual se encuentra en varias de las funciones mostradas. Este define el número de núcleos de la computadora ("cores") que se usarán para correr las diferentes funciones. Valores mayores a uno paralelizan las rutinas (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
```