ohun: an R package for optimizing automatic acoustic signal detection

Marcelo Araya-Salas [1, 2, 3] * Grace Smith-Vidaurre [4] Gloriana Chaverri [3]
Fabiola Chirino [1] Alejandro Rico-Guevara [5]

Contents

[1] Centro de Investigación en Neurociencias, Universidad de Costa Rica, San José, Costa Rica

[2] Escuela de Biología, Universidad de Costa Rica, San José, Costa Rica

[3] Sede del Sur, Universidad de Costa Rica, Golfito, Costa Rica

[4] GRACE'S AFFILIATION

[5] Department of Biology, University of Washington, Seattle, USA

* *To whom correspondence should be addressed*

Keywords:

Updated on 2022-07-06 Word count: 2951 (including chunks: 4239)

Check this link for details on how to add citations/bibliography in Rmarkdown

---

Abstract

- Description of evaluating detection performance accesible to the broader bioacoustic research community.

Introduction

Animal acoustic signals are traits widely used to investigate a variety of questions in highly diverse areas, ranging from neurobiology to community ecology and evolutionary biology. The wide usage of animal sounds in research is partly due to the fact that they can be easily registered using non-intrusive methods, can be obtained in a variety of settings from laboratories to natural areas and the equipment required for registering an analyzing this signals is increasingly inexpensive. In addition, the existence of online repositories and growing number has allowed facilitated the reseach of animal acoustic signals at larger taxonomic and geographic scales. As a result, a growing variety of a computational tools for the analysis of acoustic features on those signals is increasingly available, reflecting the growing relevance of bioacustics in the scientific tool kit.

Bioacoustic research can easily generate large amounts of data, which sometimes may proof challenging to analyze in a timely manner. Annotations that require precise location of signals in frequency and time, most commonly used in behavioral and phenotypic evolution research, remain among the most time consuming phases of analysis for this type of data. Fortunately, several tools have been develop to assist researchers in this matter. Indeed, new methods for automatic annotation of acoustic signals are continuously developed. This growing availability of tools, particularly as free software, is expected to further simplify acoustic data processing, making it accessible to a wider range of users and scientific questions. However, this diversity of tools posits a challange as it can also be hard to navigate. given the little use of widespread metrics that can evaluate the behavior of tools under different scenarios.

Here we present the new R package ohun. The package is intended to facilitate the automatic detection of acoustic signals, providing functions to diagnose and optimize detection routines. The package makes use of reference annotations containing the time position of target signals in a training data set to evaluate the performance of detection routines. This can be done with routine outputs imported from other software as well as detection run within the package itself. The package also offers an implementation of two automatic detection methods commonly used in bioacustic analysis: energy-based detection and template-based detection. We first explain how acoustic signal detection routines can be evaluate and then showcase the package usage with study cases on zebra-finch songs (*Taenopygia gutata*) and Spix's disc-winged bats (*Thyroptera tricolor*) which represent different recording settings (lab and flight cages) and signal types (sonic mating signals and ultrasonic social calls).

Diagnosing detection performance

The package makes use of signal detection theory indices to evaluate detection performance. Signal detection theory deals with the process of recovering signals (i.e. target signals) from background noise (not necessarily acoustic noise) and it's widely used for optimizing this decision making process in the presence of uncertainty. During a detection routine, the detected 'items' can be classified into 4 classes: true positives (TPs, target signals correctly identified as signal), false positives (FPs, noise incorrectly identified as 'signal'), false negatives (FNs, signals incorrectly identified as noise) and true negatives (TNs, background noise correctly identified as noise). However, TNs cannot always be easily defined in the context of acoustic signal detection as noise cannot always be partitioned in discrete units. Hence, the package makes use of TPs, FPs and FNs to calculate three additional indices that can further assist with evaluating the performance of a detection routine.

- Recall: correct detections relative to total detections (a.k.a. true positive rate or sensitivity; TPs / (TPs + FNs))

- Precision: correct detections relative to total detections (TPs / (TPs + FPs)).
- F1 score: combines recall and precision as the harmonic mean of these two, providing a single value for evaluating performance (a.k.a. F-measure or Dice similarity coefficient).

A perfect detection will have no false positives or false negatives, which will result in both recall and precision equal to 1. However, perfect detection cannot always be reached and some compromise between detecting all target signals plus some noise (recall = 1 & precision < 1) and detecting only target signals but not all of them (recall < 1 & precision = 1) might be warranted. The right balance between these two extremes will be given by the relative costs of missing signals and mistaking noise for signals, given the specific goals of the study. These indices provide an useful framework for diagnosing and optimizing the performance of a detection routine.

The package offers tools to evaluate the performance of an acoustic signal detection based on the indices described above. To accomplish this, annotations derived from a detection routine are compared against a reference table containing the time position of all target signals in the sound files. For instance, the following code evaluates a routine run in Raven Pro 1.6 (XXXX) using the "band limited energy detector" option (minimum frequency: 0.8 kHz; maximum frequency: 22 kHz; minimum duration: 0.54989 s; maximum duration: 0.54989s; minimum separation: 0.02268 s) on a subset of the Zebra finch recordings described below:

```
diagnose_detection(reference = manual_ref_tae, detection = raven_detec)
```

```
##   true.positives false.positives false.negatives split.positives
## 1            590             189               1              11
##   merged.positives overlap.to.true.positives   recall precision f1.score
## 1               76                  0.909571 0.998308  0.757381 0.861314
```

The output shows the indices described above. The function also allows to detail those indices by sound file. Here we show the first 6 files:

```
diag_raven <- diagnose_detection(reference = manual_ref_tae, detection = raven_detec, by.sound.file = TR

head(diag_raven)
```

```
##                              sound.files true.positives false.positives
## 1   Ag13_43421.27975590_11_17_7_46_15.wav             35              17
## 2    BRN7_43435.27985312_12_1_7_46_25.wav             51              36
## 3 Blk109Brn_43559.32349131_4_4_8_59_9.wav             34               2
## 4   DB118_43568.33139566_4_13_9_12_19.wav             19               2
## 5  DB15HP_43450.29217192_12_16_8_6_57.wav             21               7
## 6     DB7_43357.58119484_9_14_16_8_39.wav             19               2
##   false.negatives split.positives merged.positives overlap.to.true.positives
## 1               0               0               2                  0.988049
## 2               0               0               9                  0.969082
## 3               0               0               7                  0.906882
## 4               0               0               1                  0.987358
## 5               0               0               0                  0.988815
## 6               0               0               2                  0.933216
##   recall precision f1.score
## 1      1 0.1155116 0.207101
## 2      1 0.0688259 0.128788
## 3      1 0.9189189 0.957746
## 4      1 0.5277778 0.690909
```

```
## 5      1 0.3000000 0.461538
## 6      1 0.3653846 0.535211
```

Diagnostics from routines using different tuning parameters can be used to identify the parameter values that optimize a detection. The process of evaluating different routines for detection optimization is incorporated into the two signal detection approaches provided natively by ohun, which we depict in the following section. Note that the detection with Raven Pro does not necessarily reflect the best performance of this software and has been included only as an example on evaluating detection from external sources rather than a direct comparison of performance to ohun.

Signal detection with ohun

The package offers two methods for automatic signal detection: template-based and energy-based detection. These methods are better suited for highly stereotyped or good signal-to-noise ratio (SNR) signals, respectively. If the target signals do not fit these requirements, more elaborated methods (i.e. machine learning approaches) are warranted.

Study cases

Template detection on ultrasonic social calls of Spix's disc-winged bats

We recorded 30 individuals of Spix's disc-winged bats at Baru Biological Station, in south-western Costa Rica in January 2020. Bats were captured at their roosting sites (furled leaves of Zingeberaceae plants). Each individual bat was released in a large flight cage (9 x 4 x 3 m) for a 5 min period and their ultrasonic inquiry calls were recorded using a condenser microphone (CM16, Avisoft Bioacoustics, Glienike/Nordbahn, Germany) through an Avisoft UltraSoundGate 116Hm plugged into a laptop computer running Avisoft-Recorder software. Recordings were made at a sampling rate of 500 kHz and an amplitude resolution of 16 bits.

Recordings were manually annotated using Raven Pro 1.6 (XXXX). Annotations were created by visual inspection of spectrograms, in which the start and end of signals were determined by the location of the continuous traces of power spectral entropy of the target signals. A total of 644 calls were annotated (~21 calls per recording). Annotations were made with a time window of 200 samples and 70% of overlap. Annotations were then imported into R using the package Rraven (XXXX).

Inquiry calls of Spix's disc-winged bats are structurally stereotyped. Most variation is found between individuals although the basic form of a short, downward broadband frequency modulation is always shared (Fig. BAT-SPECTRO, Araya-Salas et al 2021 ontogeny).

Template-based detection is a useful approach when there is little structural differences in the target signals. We used this approach in ohun to detect inquiry calls. To do this, we tested the performance of three acoustic templates on a training subset of five sound files. The function `get_templates` finds several signals representative of the variation in signal structure. This function measures several spectrographic parameters which are then summarized using Principal Component Analysis. The first two components are used to project the acoustic space. On this space the function defines sub-spaces as equal-size slices of a circle centered at the centroid of the acoustic space. Templates are then selected as those closer to the centroid within each of the sub-spaces, including the centroid for the entire acoustic space. The user needs to define the number of sub-spaces in which the acoustic space will be split.

```
# read manual annotations
manual_ref_thy <- read.csv("manual_annotations_thyroptera.csv")

# get random subset of 5 sound files for training
```

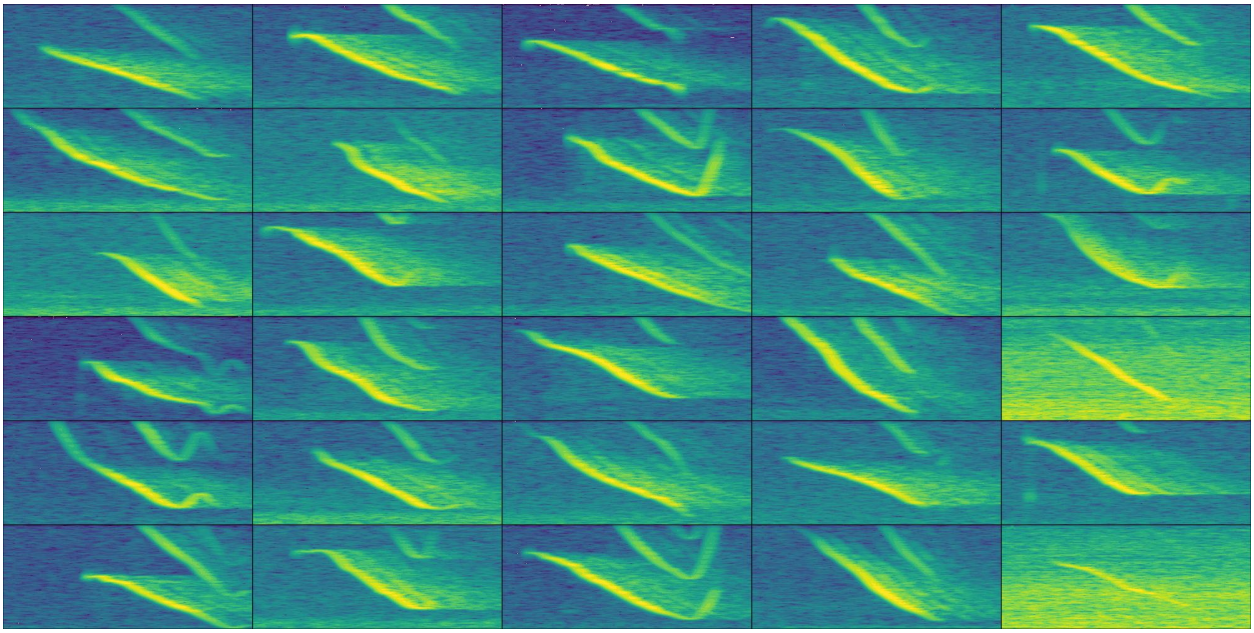

Figure 1: Fig. BAT-SPECTRO. Example spectrograms of Spix's disc-winged social calls for each of the 30 recordings used in the analysis. The highest signal-to-noise ratio call by sound file are shown. The time scale range is 0.071 s and the frequency range 10-44 kHz

```
set.seed(1)
train_files <- sample(unique(manual_ref_thy$sound.files), size = 5)
train_ref <- manual_ref_thy[manual_ref_thy$sound.files %in% train_files, ]

# the rest for testin g
test_files <- setdiff(manual_ref_thy$sound.files, train_files)
test_ref <- manual_ref_thy[manual_ref_thy$sound.files %in% test_files, ]

# find templates
templates <- get_templates(train_ref, path = data_path,  bp = c(10, 50), ovlp = 70, wl = 200, n.sub.spa
```

```
## The first 2 principal components explained 0.51 of the variance
```

The output of the `get_templates` includes an acoustic space plot (FIG. ACOUSTIC-SPACE) in which the position of the signals selected as templates is highlighted. In the following code we used those templates for detecting calls. The code iterates a template-based detection on the training data set across a range of correlation thresholds, in order to find the combination of threshold and template with the best performance.

```
# get correlation vectors
corr_templ_train <- template_correlator(
  templates = templates,
    path = data_path,
  files = unique(train_ref$sound.files),
  hop.size = 10,
  ovlp = 70
  )
```

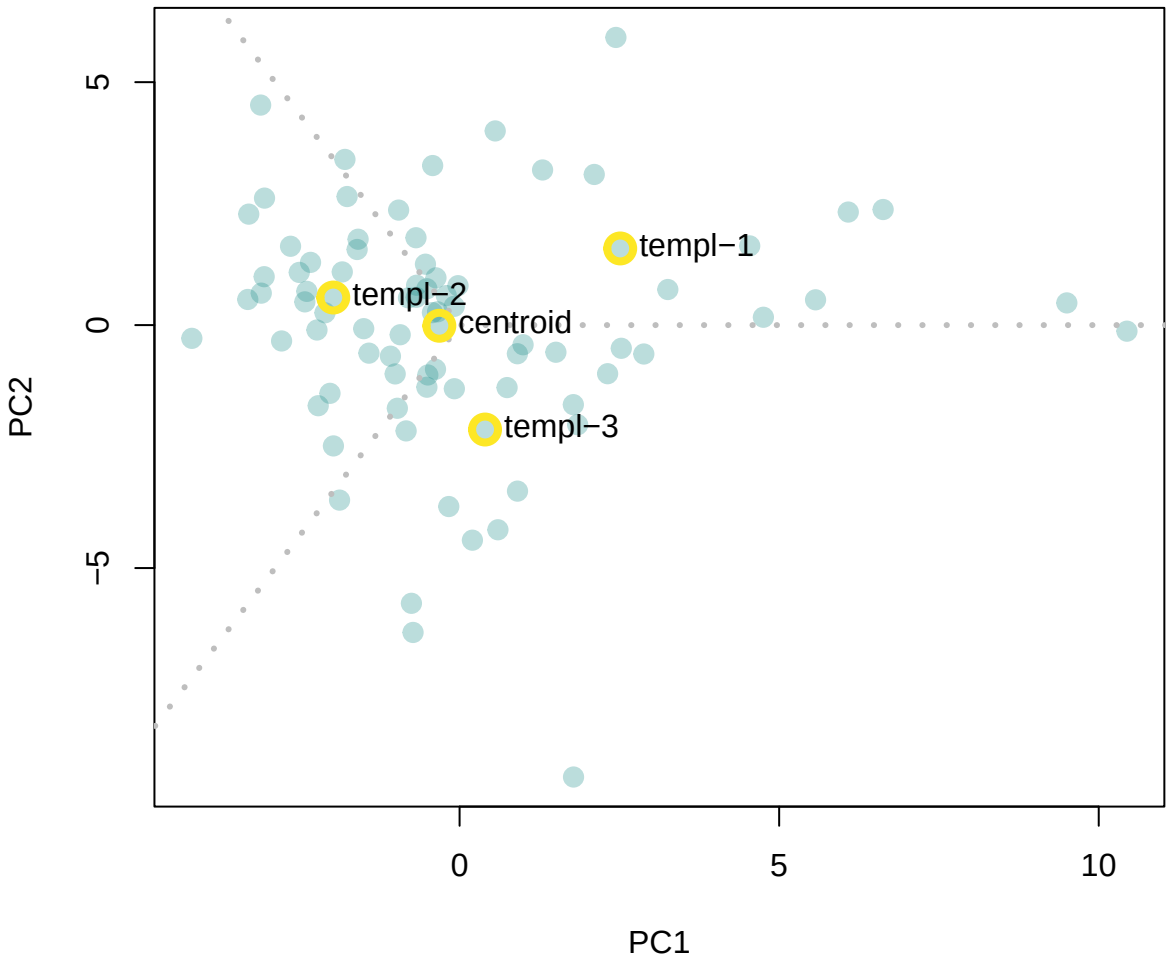

Figure 2: FIG. ACOUSTIC-SPACE. Acoustic space defined as the the first two component of a Principal Component Analysis on spectrographic parameters. Templates are selected as those closer to the centroid for each of the sub-spaces. Gray dashed lines delimit the region of sub-spaces. Yellow circles around points highlight the position of the signals selected as templates.

Table 1: TABLE TEMPLATE PERFORMANCE. Performance diagnostic of template-based detections using four templates across several threshold values. Only the two highest performance iterations for each template are shown.

threshold	templates	true.positives	false.positives	false.negatives	recall	precision	f1.score
0.45	centroid	81	1	0	1.000000	0.987805	0.993865
0.50	centroid	80	0	1	0.987654	1.000000	0.993789
0.50	templ-1	77	1	4	0.950617	0.987179	0.968553
0.45	templ-1	80	5	1	0.987654	0.941176	0.963855
0.40	templ-2	79	0	2	0.975309	1.000000	0.987500
0.35	templ-2	80	2	1	0.987654	0.975610	0.981595
0.45	templ-3	77	2	4	0.950617	0.974684	0.962500
0.40	templ-3	79	5	2	0.975309	0.940476	0.957576

```
# evaluate detection for different correlation thresholds
opt_detec_train <- optimize_template_detector(
  reference = train_ref,
  template.correlations = corr_templ_train,
  threshold = seq(0.05, 0.5, 0.01)
  )
```

Note that the correlation vectors are estimated first (i.e. vectors of correlation values across sound files, `template_correlator()`) and then the correlation thresholds are optimized on them (`optimize_template_detector()`). The output of `optimize_template_detector()` contains the detection performance indices for each combination of templates and thresholds. Table TEMPLATE PERFORMANCE shows the two highest performance runs for each template.

We can explore the performance of each template in more detail by looking at the change in F1 score across thresholds (FIG. THRESHOLD vs F1.SCORE).

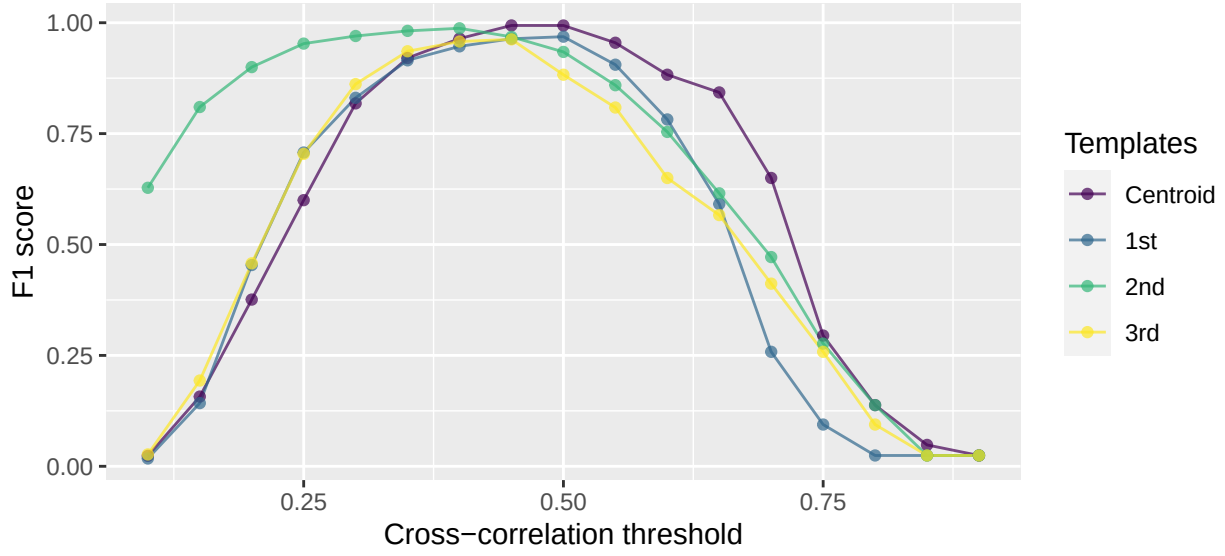

Figure 3: FIG. THRESHOLD vs F1.SCORE. Shows the changes in F1 score across the range of threshold values

In this example the "centroid" template, produced the best performance (TABLE TEMPLATE PERFOR-

MANCE; FIG. THRESHOLD vs F1.SCORE). Hence, we will use this template for detecting calls on the rest of the data. The following code extracts this template from the reference annotation table and use it to find inquiry calls on the testing data set:

```
# get correlation vectors for test files
corr_templ_test <- template_correlator(
  templates = templates[templates$sound.file == "centroid", ],
    path = data_path, files = unique(test_ref$sound.files),
  hop.size = 10,
  ovlp = 70
  )

# detect on test files
detec_test <- template_detector(
  template.correlations = corr_templ_test,
    threshold = 0.45
  )

diagnose_detection(reference = test_ref, detection = detec_test)
```

```
##   true.positives false.positives false.negatives split.positives
## 1            536              10              27               2
##   merged.positives overlap.to.true.positives   recall precision f1.score
## 1               16                 0.906031 0.952043  0.981685 0.966637
```

The last line of codes evalutes the detection on the test data set, which shows a good performance for both recall and precision.

Energy-based detection on zebra finch songs

We used recordings from 18 zebra finch males. These recordings were obtained ....

Zebra finch vocalizations are composed by multiple elements (i.e. distinct patterns of continuous traces of power spectral entropy in the spectrogram separated by time gaps) that can vary importantly in key features as duration and frequency range (ZEBRAFINCH-SPECTRO). However, as the recorded signals show a good signal-to-noise ratio, they can potentially be detected using an energy-based approach.

The following code loads the reference annotations and split them in two data set for training (3 sound files) and testing (15 sound files):

```
manual_ref_tae <- read.csv("manual_selections_Taeniopygia.csv")

set.seed(450)
train_files <- sample(unique(manual_ref_tae$sound.files), 3)
test_files <- setdiff(manual_ref_tae$sound.files, train_files)

train_ref <- manual_ref_tae[manual_ref_tae$sound.files %in% train_files, ]
test_ref <- manual_ref_tae[manual_ref_tae$sound.files %in% test_files, ]
```

Now we can optimize the detection parameters using the function `optimize_energy_detector`. This function runs a detection for all possible combinations of tuning parameters. The code below tries three minimum duration and maximum duration values and two hold time values (which merges signals within the specified time interval):

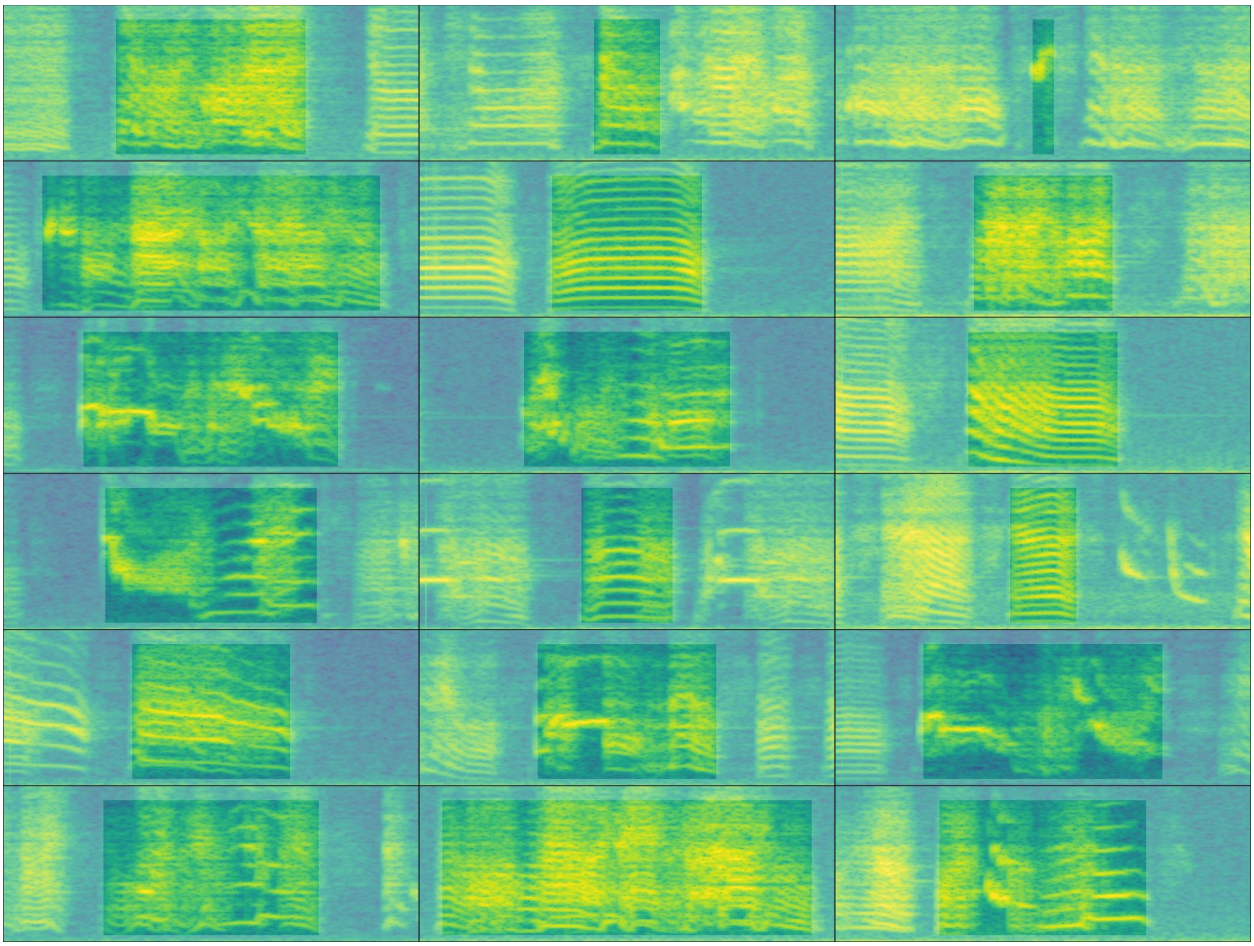

Figure 4: Fig. ZEBRAFINCH-SPECTRO. Example spectrograms of male zebra finch songs for each of the 18 sound files used in the analysis. The highest signal-to-noise ratio call by sound file are shown. The time scale range is 0.359 s and the frequency range 0-11 kHz. Signals have been highlighted for visualization purposes only.

```
feature_reference(test_ref)
```

```
##                 min   mean     max
## sel.duration 23.42 102.18  319.41
## gap.duration 80.19 355.84 3024.23
## bottom.freq   0.50   0.50    0.50
## top.freq     10.00  10.00   10.00
```

```
opt_det_train <- optimize_energy_detector(
  reference = train_ref,
  files = train_files,
  threshold = c(1, 5),
  hop.size = 11.6,
  smooth = c(5, 10),
  hold.time = c(0, 5),
  min.duration = c(5, 15, 25),
  max.duration = c(275, 300, 325),
  bp = c(0.5, 10)
)
```

The output (`opt_det_train`) shows the performance indices for each of those combinations, here we show the 10 combinations with the highest F1 score:

```
# subset with highest performance
opt_det_train <- opt_det_train[order(opt_det_train$f1.score, decreasing = TRUE), ]

head(opt_det_train, 10)
```

```
##    threshold smooth hold.time min.duration max.duration true.positives
## 45         1      5         5           25          300            105
## 69         1      5         5           25          325            105
## 41         1      5         0           25          300            105
## 65         1      5         0           25          325            105
## 43         1     10         0           25          300            105
## 67         1     10         0           25          325            105
## 37         1      5         5           15          300            105
## 61         1      5         5           15          325            105
## 14         5      5         5           15          275             96
## 38         5      5         5           15          300             96
##    false.positives false.negatives   recall precision f1.score
## 45               9               0 1.000000  0.921053 0.958904
## 69               9               0 1.000000  0.921053 0.958904
## 41              12               0 1.000000  0.897436 0.945946
## 65              12               0 1.000000  0.897436 0.945946
## 43              13               0 1.000000  0.889831 0.941704
## 67              13               0 1.000000  0.889831 0.941704
## 37              15               0 1.000000  0.875000 0.933333
## 61              15               0 1.000000  0.875000 0.933333
## 14               8               9 0.914286  0.923077 0.918660
## 38               8               9 0.914286  0.923077 0.918660
```

We now can use the tuning parameter values that produce the best performance to detect signals on the test data set:

```
best_param <- opt_det_train[which.max(opt_det_train$f1.score), ]

det_test <- energy_detector(
  files = test_files,
  threshold = best_param$threshold,
  hop.size = 11.6,
  smooth = best_param$smooth,
  hold.time = best_param$hold.time,
  min.duration = best_param$min.duration,
  max.duration = best_param$max.duration,
  bp = c(0.5, 10)
)
```

As our reference annotations include all sound files we can evaluate the performance of the detection on the test set as well:

```
diagnose_detection(reference = test_ref, detection = det_test, by.sound.file = FALSE)
```

```
##   true.positives false.positives false.negatives split.positives
## 1            475              27              11               4
##   merged.positives overlap.to.true.positives   recall precision f1.score
## 1               16                  0.969641 0.977366  0.946215 0.961538
```

The performance on the test data set was also acceptable with a F1 score of 0.95. Note that in the example we use a small subset of sound files for training. More training data might be needed for optimizing a detection routine on larger data sets or recordings with more variable signals or background noise levels.

Additional tools

The package offers additional tools to simplify acoustic signal detection. The function 'feature_reference' allows to extract information about the duration and frequency range of acoustic signals than can then be used to define tuning parameter values. This can be particularly useful for energy-based detection in which time and frequency attributes of the target signals are required. Detected signals can be labeled as false or true positives with the function 'label_detection.' This allows users to explore the structure of false positives and figure out ways to exclude them. The function 'filter_detections' can be used to remove ambiguous signals (i.e. those labeled as split or merged detection) keeping only those that maximize a specific criterium (i.e. the highest template correlacion). Finally, note that several templates representing the range of variation in signal structure can be used to detect semi-stereotyped signals when running template-based detection ('template_detection' function).

Discusion

Here we have showed how to evaluate the performance of acoustic signal detection routines using the package ohun. The package can evaluate detection outputs imported from other software, as well as its own detection routines. The latter can be iterated over combinations of tuning parameters in order to find those values that optimize a detection. Despite signal detection indices being commonly reported when presenting new automatic detection methods, to our knowledge widely applicable performance evaluating routines have not been made available in a free, open source platform. Providing a common framework for acoustic signal detection evaluation can simplify comparing performances of different tools and the selection of those better

suited to our system and research question. Note that the tools offered by ohun for diagnosing a detection should not be necessarily limited to acoustic data. Any type of automatic detection in which the time of occurrence of discrete events needs to be found can be evaluated and optimized by comparing it to a reference annotation. For instance detection of specific behaviors in video analysis of animal motor activity (Sturman et al 2020; Hsu & Yttri 2021; DeepEthogram).

The package provides two native detection methods: template-based and energy-based detection. Compared to new deep learning approaches for finding the occurrence of acoustic signals, the two native methods are relatively simple tools. However, these methods have been widely used by the bioacoustic community and, under the appropriate conditions can reach adequate performance, as we have shown in our two study cases. Deep learning methods tend to require larger computational power and more complex training routines. This might bring unnecessary difficulties when dealing when less challenging detection tasks. Therefore the availability of a wide range of approaches is desired in order to simplify finding the most appropriate tool for the intricacies of our study system and research goals. Note that the tools offered in ohun can be used in a pipeline in which detected signals are further classified using more elaborated discrimination algortihms. Acoustic parameters can be used to quantify the structure of the signals and tell apart target from non-target signals.

Detection routines can take a long time when working with large amounts of acoustic data (e.g. large recordings and/or many files). These are some tips that can help make a routine more time-efficient. 1. Always test procedures on small data subsets. Make sure you are getting decent results on a small subset of recordings before trying to scale up the analysis. 2. Template-based detection is almost always faster than energy-based detection. 4. Run routines in parallel. Parallelization (i.e. the ability to distribute tasks over several cores in your computer) can significantly speed-up routines. All functions for automatic detection and performance evaluation allow users to run analysis in parallel (see `parallel` argument in those functions). Hence a computer with several cores can be helpful for improving efficiency. 4. Sampling rate matters. Detecting signals on low sampling rate files is faster, so we must avoid having Nyquist frequencies much higher than the highest frequency of the target signals. 5. Try using a computer with lots of RAM memory or a computer cluster for working on large amounts of data. These rules are not restricted to ohun and can also be helpful to speed-up routines in other software packages.

There are some additional things to be considered when aiming to automatically detect acoustic signals. When running automatic signals detection try to use your knowledge about the signal structure to determine the initial range. This can be extremely helpful for narrowing down possible parameter values, particularly for energy-based detection. As a general rule, if people have a hard time figuring out where a target signal occurs, detection algorithms will also have a hard time. In cases in which occurrences are ambiguous low performances are expected. Make sure reference tables contain all target signals and only the target signals, otherwise performance optimization can be misleading as the performance of a detection cannot be better than the reference itself. Lastly, avoid having overlapping signals or several signals as a single one (like a multi-syllable vocalization) in the reference table when running an energy-based detector, as they are likely to be identified as separeted units.

https://cran.r-project.org/web/packages/bioacoustics/vignettes/tutorial.html

References

Sturman, O., von Ziegler, L., Schläppi, C. et al. Deep learning-based behavioral analysis reaches human accuracy and is capable of outperforming commercial solutions. Neuropsychopharmacol. 45, 1942–1952 (2020). https://doi.org/10.1038

Hsu, A.I., Yttri, E.A. B-SOiD, an open-source unsupervised algorithm for identification and fast prediction of behaviors. Nat Commun 12, 5188 (2021). https://doi.org/10.1038/s41467-021-25420-x