Package: whatsappaudioR (via r-universe)

November 1, 2026

Title Convert Compressed Mobile Voice Notes into Tidy Spectrogram Data

Version 0.2.0

URL https://github.com/jitu-droid/sonicscout

BugReports https://github.com/jitu-droid/sonicscout/issues

Description Reads compressed '.opus'/'.ogg' voice notes -- the kind produced by WhatsApp and similar VoIP messaging apps -- and converts them directly into tidy data frames of spectrogram and spectral-feature data, ready for analysis or modelling. Also provides a function to simulate the lossy compression such messaging apps apply, for testing how well a signal survives it. Built for a bioacoustic pest-detection pipeline (SonicScout) but has no dependency on that domain.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

Depends R (>= 4.1)

SystemRequirements FFmpeg (with libopus)

Imports tuneR, seewave, tibble, stats, tools

Suggests testthat (>= 3.0.0), knitr, rmarkdown, ggplot2

VignetteBuilder knitr

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

Config/pak/sysreqs libsndfile1-dev

Repository https://jitu-droid.r-universe.dev

Date/Publication 2026-11-01 16:16:55 UTC

RemoteUrl https://github.com/jitu-droid/sonicscout

RemoteRef HEAD

RemoteSha 8383d2a12b4b43799b4a07ba23ecbdf62edcaf16

RemoteSubdir pkg/whatsappaudioR

Contents

wa_batch 2
wa_features 3
wa_read 4
wa_simulate_compression 5
wa_spectrogram 6

Index **7**

`wa_batch` *Vectorise a function over every audio file in a directory*

Description

Reads every supported audio file under `dir` (recursively) and applies `fn` (default `wa_features()`) to each, row-binding the results with a leading `file` column identifying the source path.

Usage

```
wa_batch(dir, fn = wa_features, pattern = "\\.(opus|ogg|wav|mp3|m4a)$", ...)
```

Arguments

`dir`	Directory to scan for audio files.
`fn`	Function taking a tuneR::Wave object and returning a single-row data frame/tibble. Defaults to `wa_features()`.
`pattern`	Regex for filenames to include. Defaults to the extensions `wa_read()` supports.
`...`	Passed on to `fn`.

Value

A tibble: one row per file, with a leading `file` column, plus whatever columns `fn` returns.

Examples

```
## Not run:
features_df <- wa_batch("voice_notes/", wa_features)

## End(Not run)
```

`wa_features` *Compute a one-row tibble of spectral features from a Wave object*

Description

Extracts a documented, inspectable feature set rather than a black-box embedding, so results can be explained rather than just trusted: `peak_freq_hz`, `spectral_centroid_hz`, `spectral_flatness`, `spectral_entropy`, `spectral_rolloff85_hz` (frequency below which 85% of spectral energy lies), `dominant_freq_sd_hz` (stability of the dominant frequency contour over time – low means tonal/steady, high means erratic), `harmonic_energy_ratio` (a simple heuristic, see Details), and 12 MFCCs (mean over frames).

Usage

```
wa_features(wave, wl = 1024)
```

Arguments

`wave`	A tuneR::Wave object, e.g. from `wa_read()`.
`wl`	Window length (samples) used for the underlying STFT/mean spectrum. See `wa_spectrogram()` for the same parameter.

Details

`harmonic_energy_ratio` is the fraction of total spectral energy that falls within a narrow band around the first 5 integer multiples of the peak frequency. It is a lightweight, transparent heuristic for "how harmonic does this sound look", not a validated harmonicity/HNR index – report it as such.

Value

A one-row tibble.

Examples

```
## Not run:
w <- wa_read("voice_note.opus")
wa_features(w)

## End(Not run)
```

`wa_read` *Read a compressed voice-note audio file into a Wave object*

Description

Handles the container/codec mess of mobile voice notes: `.opus` and `.ogg` files are decoded via a system `ffmpeg` binary into a temporary WAV before being read with `tuneR::readWave()`. `.wav` files are read directly, and `.mp3`/`.m4a` are routed through the same decode step as opus/ogg.

Usage

```
wa_read(path, sample_rate = NULL, mono = FALSE)
```

Arguments

`path`	Path to an audio file (`.opus`, `.ogg`, `.wav`, `.mp3`, `.m4a`).
`sample_rate`	Optional target sample rate in Hz. `NULL` (default) keeps the file's own rate.
`mono`	If `TRUE`, downmix to one channel (averaging channels). Default `FALSE` keeps the file's channels.

Value

A tuneR::Wave object.

WORKAROUND: decoding via system ffmpeg instead of `av`

The obvious implementation would use `av::av_audio_convert()` (`av` binds FFmpeg's libav* libraries directly, no subprocess needed). That was tried first, but on the machine this package was built and tested on, `av::av_audio_convert()` and even `av::av_media_info()` segfault the R process outright – confirmed even against `av`'s own bundled sample file, so it isn't an input-file issue. A segfault can't be caught with `tryCatch()`, which makes `av` unsafe to call from a long-running process (a webhook server, a Shiny session) on that machine. See `BLOCKERS.md` in the parent repo for the full diagnosis. This function shells out to a system `ffmpeg` binary instead, which is confirmed working. If a future `av` version/rebuild is confirmed not to segfault on your machine, switching back to `av::av_audio_convert()` would avoid the subprocess-spawn overhead.

Sample rate

By default the audio comes back at whatever rate the file decodes to – 48 kHz for Opus, usually 44.1 or 48 kHz for MP3/WAV. A model trained at one sample rate must be fed that same rate: spectral features computed over a different bandwidth are not comparable. Pass `sample_rate` (and `mono = TRUE`) to have ffmpeg resample on decode, using its filtered resampler (with anti-aliasing), not naive decimation.

Examples

```
## Not run:
w <- wa_read("voice_note.opus")
w16 <- wa_read("voice_note.opus", sample_rate = 16000, mono = TRUE)

## End(Not run)
```

`wa_simulate_compression` *Simulate lossy VoIP voice-note compression on a Wave object*

Description

Round-trips a tuneR::Wave object through the same codec pipeline WhatsApp (and similar apps) apply to voice notes: resample to `sr` Hz, downmix to mono, encode to Opus at `bitrate` bits/sec inside an Ogg container, then decode back to PCM.

Usage

```
wa_simulate_compression(wave, bitrate = 16000, sr = 16000)
```

Arguments

`wave`	A tuneR::Wave object.
`bitrate`	Target bitrate in bits/sec (default `16000`, i.e. 16 kbps, matching WhatsApp's voice-note encoding).
`sr`	Target sample rate in Hz (default `16000`, matching WhatsApp).

Value

A tuneR::Wave object: the degraded, round-tripped audio.

WORKAROUND: both encode and decode go through system ffmpeg

`av::av_audio_convert()` would be the obvious choice here, and its installed version does have a `bit_rate` argument – but empirically it (and even `av::av_media_info()`) segfaults R outright on the machine this package was built and tested on, including against `av`'s own bundled sample file. A segfault can't be caught with `tryCatch()`, which makes it unsafe to call from a long-running process. See `BLOCKERS.md` in the parent repo. Both the encode and decode steps here shell out to a system `ffmpeg` binary (with `libopus`) instead. This requires `ffmpeg` to be on `PATH`; call `Sys.which("ffmpeg")` to check.

Examples

```
## Not run:
w <- wa_read("clean_recording.wav")
degraded <- wa_simulate_compression(w, bitrate = 16000, sr = 16000)

## End(Not run)
```

`wa_spectrogram` *Compute a tidy spectrogram data frame from a Wave object*

Description

Wraps `seewave::spectro()` and reshapes its matrix output into a long tibble of `time` / `freq` / `amplitude`, which composes far more easily with `dplyr`/`ggplot2` than the raw list-of-matrices `seewave` returns.

Usage

```
wa_spectrogram(wave, wl = 1024, ovlp = 50)
```

Arguments

`wave`	A tuneR::Wave object, e.g. from `wa_read()`.
`wl`	Window length (samples) for the STFT. Longer windows give better frequency resolution at the cost of time resolution – for low-frequency signals (like insect wingbeats) prefer a longer window (e.g. 1024-2048) over the seewave default.
`ovlp`	Percent overlap between successive analysis windows.

Value

A tibble with columns `time` (seconds), `freq` (kHz, matching `seewave`'s convention), and `amplitude`.

Examples

```
## Not run:
w <- wa_read("voice_note.opus")
spec_df <- wa_spectrogram(w, wl = 2048)

## End(Not run)
```

Index

av::av_audio_convert(), *4, 5*

seewave::spectro(), *6*

tuneR::readWave(), *4*
tuneR::Wave, *2–6*

wa_batch, 2
wa_features, 3
wa_features(), *2*
wa_read, 4
wa_read(), *2, 3*, *6*
wa_simulate_compression, 5
wa_spectrogram, 6
wa_spectrogram(), *3*