PYSOX: LEVERAGING THE AUDIO SIGNAL PROCESSING POWER OF SOX IN PYTHON

Rachel M. Bittner[1,2], **Eric Humphrey**[2], **Juan P. Bello**[1]
1 Music and Audio Research Lab, New York University
2 Spotify Inc.

ABSTRACT

SoX is a popular command line tool for sound processing. Among many other processes, it allows users to perform a repeated process (e.g. file conversion) over a large batch of audio files and apply a chains of audio effects (e.g. compression, reverb) in a single line of code. SoX has proven to be a useful resource for Music Information Retrieval (MIR) tasks, and in particular for dataset creation. While the library is powerful and stable, building long strings of command line arguments can be messy and error prone. We present `pysox`, a Python library that provides a simple interface between Python and SoX, making it easier to incorporate SoX into MIR workflows.

1. INTRODUCTION

The SoX (Sound exchange) library [3] is a widely used cross-platform tool for sound processing. The first version of SoX was first released in 1991 [1], and is still actively maintained. With over 25 years of development, SoX is both powerful and stable. Self-branded as "the swiss army knife of sound processing," it performs a huge range of useful operations, such as file format conversions, cutting and splicing, silence removal, and more complex effects such as reverb and compression. Among many other projects, it powers the popular open-source audio editing application *Audacity* [1]. Unlike most audio processing software, SoX has a wide array of codec support, and can thus read and convert between virtually every known audio file format [2].

Developing MIR algorithms frequently requires batch processing of audio files. Tasks like format normalization, splitting files into small segments, or adding background noise are tedious and error prone when done by hand, but can be performed in one line of code with SoX. For example, SoX was heavily utilized in the creation of several audio datasets including MedleyDB [4, 5] and the Urban Sound datasets [11].

While the SoX library is powerful, it requires careful parsing of the documentation and is non-trivial to master. For example, applying compression to the file `input.wav` and save it as `output.wav` could be achieved with the following command:

```
$ sox input.wav output.wav compand \
    0.3,0.8 6.0:-70,-70,-60,-20,0,0
```

Understanding the meanings of each of the numbers following the `compand` argument requires digging into the documentation, and the command cannot be used without the string of numbers, as there is no default setting. Furthermore, when multiple effects are applied, the commands quickly become long and unreadable. Other command-line audio processing tools such as `ffmpeg` [2] suffer from similar challenges.

Building upon SoX, we present `pysox`, a software library that provides a user-friendly Python interface to the commandline utility. With many modern MIR workflows being written in Python, `pysox` helps bridge the gap between this powerful utility and research code, providing additional functionality for argument and improved debugging. Whenever possible, we provide reasonable default settings for effects and documentation for what each of the arguments controls. Note that the decision to wrap the commandline interface is motivated by two realities. While it may be tempting to write a pure-Python library that has similar functionality to SoX, wrapping compiled routines is more efficient to develop and execute, and benefits from the stability and wisdom of the existing library. Alternatively, the software license of SoX is not conducive to the development of C-bindings, like the OpenCV project [3], being released under the GPLv2 licence [4], which would otherwise handcuff its widespread usage.

There have recently been a number of python tools developed for music and audio processing, such as `librosa` [8], `essentia` [6], `mir_eval` [10], `muda` [7], or `pretty-midi` [9], and `pysox` complements the functionality of these tools. For example, `pysox` can be used for standardizing audio formats for use with these libraries, or for efficiently applying trimming, fades, or compression to a collection of audio.

[1] `http://sox.sourceforge.net/SoX/History`

[2] Available when the appropriate add-ons are installed, such as liblame support for mp3s or libvorbis for oggs.

[3] `http://docs.opencv.org/`

[4] `https://sourceforge.net/directory/os:mac/license:gpl/`

The pysox library is released open-source on github [5] and can be installed with pip:

```
$ pip install sox
```

2. FUNCTIONALITY

The functionality supported in pysox includes *transformations* of single audio files, *combinations* of multiple audio files, and retrieval of audio file information.

2.1 Transformers

Transformer objects are used to apply a chain of effects to an input audio file and save the new file to a specified output path.

The most basic usage of a Transformer object is to convert a file to a different format:

```
>>> tfm = sox.Transformer(
    'path/to/input_audio.wav',
    'path/to/output/audio.aiff')
>>> tfm.build()
```

Output files are not created until the build command is called.

Any number of effects can be chained together. For example, the following code changes the sample rate to 8000 Hz, normalizes it to -3 dB, removes any long silences, trims the file to be 30 seconds long, adds reverb, applies a fade in/out, and saves the output as an mp3 file.

```
# initialize transformer
>>> tfm = sox.Transformer(
    'path/to/input_audio.wav',
    'path/to/output/audio.mp3')
# change sample rate to 8000 Hz
>>> tfm.rate(samplerate=8000)
# normalize to -3 dB
>>> tfm.norm(db_level=-3)
# remove any long silences
>>> tfm.silence()
# trim the audio to the first 30 seconds
>>> tfm.trim(0, 30)
# add reverb with default parameters
>>> tfm.reverb()
# apply a fade in and fade out
>>> tfm.fade(fade_in_len=1.0,
             fade_out_len=0.5)
# create the output file.
>>> tfm.build()
```

If incompatible arguments are passed to one of the effects, pysox throws an error. This is an advantage over the command line version of SoX, where errors are only thrown when attempting to execute the command; oftentimes files are still created, making it difficult to tell when something went wrong and what was responsible for it.

Transformer objects additionally have a play method, which allows the user to preview the audio rendered with the current set of effects without actually creating the audio file.

[5] https://github.com/rabitt/pysox

2.2 Combiners

Combiner objects are used to combine multiple audio files in the manner specified (e.g. mixing), optionally apply effects, and save the output file to a specified path. Multiple audio files may be combined by (1) concatenating (2) stacking into multiple channels (3) mixing (sum or power sum) or (4) multiplying. Optionally, volume adjustments can be applied to each input file before combining.

For example, AM modulation could be applied to a carrier by multiplying:

```
>>> cbn = sox.Combiner(
    ['modulator.aiff', 'carrier.mp3'],
    'AM_modulated.wav',
    combine_type='multiply')
>>> cbn.build()
```

All effects that can be applied to single files can be applied to the output of the Combiner:

```
# initialize Combiner. Mixes file2.wav
# at half its original volume.
>>> input_files = [
    'file1.wav', 'file2.wav', 'file3.mp3']
>>> cbn = sox.Combiner(
    input_files, 'output.wav',
    combine_type='mix',
    input_volumes=[1, 0.5, 1])
# low pass filter the mixed audio
>>> cbn.lowpass(frequency=2000)
# reverse the audio
>>> cbn.reverse()
# create the output file
>>> cbn.build()
```

2.3 File Information

In addition to manipulating files, SoX can also look up information about a given audio file. pysox provides this in the sox.file_info module. Information such as the duration, number of channels, sample encoding, and sample statistics can be extracted.

```
from sox import file_info
>>> file_info.duration('72sec_file.mp3')
72.0
>>> file_info.channels('stereo_file.wav')
2
>>> file_info.sample_encoding('music.ogg')
Vorbis
>>> file_info.silent('silent_file.aiff')
True
```

These methods are particularly useful for performing transformations when the state of the source audio is unknown.

3. FUTURE WORK

The core functionality of SoX is included in pysox, however there are common operations, such as crossfading, that are possible in the existing framework but tedious. We would like to expand pysox to include cookbook-style utility methods.

4. REFERENCES

[1] *Audacity*, (accessed July 31, 2016). `http://www.audacityteam.org/`.

[2] *FFmpeg*, (accessed July 31, 2016). `http://ffmpeg.org/`.

[3] *SoX Sound eXchange*, (accessed July 31, 2016). `http://sox.sourceforge.net/`.

[4] Rachel M Bittner, Justin Salamon, Mike Tierney, Matthias Mauch, Chris Cannam, and Juan Pablo Bello. MedleyDB: A multitrack dataset for annotation-intensive mir research. In *ISMIR '14 (Taipei, Taiwan)*, pages 155–160, 2014.

[5] Rachel M. Bittner, Julia Wilkins, Hanna Yip, and Juan P. Bello. MedleyDB 2.0: New data and a system for sustainable data collection. In *Proceedings of the 17th International Society for Music Information Retrieval Conference Late Breaking and Demo Papers*, 2016.

[6] Dmitry Bogdanov, Nicolas Wack, Emilia Gómez, Sankalp Gulati, Perfecto Herrera, Oscar Mayor, Gerard Roma, Justin Salamon, José R Zapata, and Xavier Serra. Essentia: An audio analysis library for music information retrieval. In *In Proceedings of the 14th International Society for Music Information Retrieval Conference, ISMIR*, pages 493–498. Citeseer, 2013.

[7] B. McFee, E.J. Humphrey, and J.P. Bello. A software framework for musical data augmentation. In *16th International Society for Music Information Retrieval Conference*, ISMIR, 2015.

[8] B. McFee, C. Raffel, D. Liang, D.P.W. Ellis, M. McVicar, E. Battenberg, and O. Nieto. librosa: Audio and music signal analysis in python. In *14th annual Scientific Computing with Python conference*, SciPy, July 2015.

[9] Colin Raffel and Daniel PW Ellis. Intuitive analysis, creation and manipulation of midi data with pretty_midi. In *Proceedings of the 15th International Society for Music Information Retrieval Conference Late Breaking and Demo Papers*, 2014.

[10] Colin Raffel, Brian McFee, Eric J Humphrey, Justin Salamon, Oriol Nieto, Dawen Liang, Daniel PW Ellis, and C Colin Raffel. mir_eval: A transparent implementation of common mir metrics. In *In Proceedings of the 15th International Society for Music Information Retrieval Conference, ISMIR*. Citeseer, 2014.

[11] J. Salamon, C. Jacoby, and J. P. Bello. A dataset and taxonomy for urban sound research. In *22st ACM International Conference on Multimedia (ACM-MM'14)*, Orlando, FL, USA, Nov. 2014.