THE NORTHWESTERN UNIVERSITY SOURCE SEPARATION LIBRARY

Ethan Manilow, Prem Seetharaman, Bryan Pardo
Northwestern University
{ethanmanilow@u., prem@u., pardo@}northwestern.edu

ABSTRACT

Audio source separation is the process of isolating individual sonic elements from a mixture or auditory scene. We present the Northwestern University Source Separation Library, or `nussl` for short. `nussl` (pronounced '*nuzzle*') is an open-source, object-oriented audio source separation library implemented in Python. `nussl` provides implementations for many existing source separation algorithms and a platform for creating the next generation of source separation algorithms. By nature of its design, `nussl` easily allows new algorithms to be benchmarked against existing algorithms on established data sets and facilitates development of new variations on algorithms. Here, we present the design methodologies in `nussl`, two experiments using it, and use `nussl` to showcase benchmarks for some algorithms contained within.

1. INTRODUCTION

Audio source separation is the process of isolating individual sonic elements from a mixture or auditory scene. The underdetermined case is where there are fewer mixture channels (e.g. a stereo recording) than sources (a string quartet). Examples of underdetermined source separation include extracting a single speaker from a single-mic recording of a crowded cocktail party, extracting a singer from a rock band recording, or removing an extraneous car horn from a field recorded interview. Applications of source separation include end-user tools for extracting vocals (e.g., Audionamix ADX Trax), upmixing vintage recordings to stereo or 5.1 surround sound, and as a preprocessing step for speech recognition [15] and other audio tasks.

There have been many approaches taken to source separation in the underdetermined case. These include Non-negative Matrix Factorization (NMF) [37, 38], harmonic/percussive separation [7], deep learning [11, 13, 14, 16, 21, 25], pitch tracking [5, 34], spatialization [8, 32], repeating vs non-repeating elements [29,30,36], low-rank vs sparse decomposition [12], and common fate [24, 39, 44].

The research community has centered around a collection of common data sets to benchmark results from these different approaches. Perhaps the best known are the data sets used for the recurring Signal Separation Evaluation Campaign (SiSEC) [19, 42]. SiSEC previously used DSD100 [20],and now uses both DSD100 and MedleyDB [1], calling the combined data set MUSDB18 [28]. Other common data sets include iKala [2], MIR-1K [3], TIMIT [10], and WSJ0 [9]. The community also typically uses the signal quality measures provided by BSS-Eval [6, 42] (SDR, SIR, and SAR) when reporting results.

Though there is some debate about this [4], it can be argued that using common data sets and evaluation measures strengthens research through standardizing metrics by making new and existing research directly comparable. While the source separation community has common data sets and common evaluation measures, there exists no such common code repository for actual implementations of proposed algorithms.

Vandewalle *et al.* [41] argue that in the computational sciences, implementation details are crucial to reproducing the results of academic papers, despite being routinely omitted from publications. They establish six degrees of reproducibility, scored from 0 (lowest) to 5 (highest). A score of 5 is defined as "The results can be easily reproduced by an independent researcher with at most 15 min of user effort, requiring only standard, freely available tools." A 0 indicates research completely unreproducible by an independent researcher.

The ubiquity of code repositories like Github has allowed many researchers to share their code, but using Github is not a guarantee of easy reproducibility. A recent seminar [1] convened to reproduce results from six MIR papers (including two source separation papers) and concluded that not a single paper, *despite including code*, scored better than "Can be reproduced, requiring considerable effort" using the reproducibility scorecard by Vandewalle *et al.* [41]. Of the two source separation papers, both scored "Could be reproduced, requiring extreme effort."

This work aims to provide a common platform for researchers to contribute their source separation algorithms to fill the implementation gap and promote reproducibility within the source separation research community. Furthermore, this work strives to make every algorithm in the proposed framework achieve the highest reproducibility rating using the Vandewalle *et al.* scorecard: reproducible results

[1] https://github.com/audiolabs/APSRR-2016

in under 15 minutes. The Northwestern University Source Separation Library (`nussl`) is the culmination of that effort.

In this paper we will explore `nussl` and introduce some core aspects of its design methodology, provide an outline about how to add a new algorithm to `nussl` and benchmark many of the source separation algorithms in `nussl`. We also leverage the flexibility of the `nussl` framework to implement and test novel combinations of existing source separation algorithms. We examine how source separation algorithms interact with methods such as overlap and add, which apply the same source separation algorithm to overlapping windows in the mixture and recombine the sources afterwards, rather than applying them to the entire mixture. More information about `nussl` can be obtained at the project's online documentation. [2] A companion website is also provided for this paper. [3]

2. RELATED WORK

The biennial Signal Separation Evaluation Campaign (SiSEC) [19,42] is an open call for members of the MIR research community to submit source separation algorithms to be run and evaluated on a common dataset. While the dataset is widely distributed and used, not all of the code submissions from previous campaigns have been made available to scrutinize. Additionally, SiSEC offers no standard API to adhere to, and only a minimal framework to work with. We have submitted many algorithms within `nussl` to the most recent SiSEC campaign.

Other source separation libraries have been presented in the past, as well. The *Flexible Audio Source Separation Toolbox* (FASST) [23, 35] [4] was written in MATLAB and C++, but did not have a process for outside submissions. `untwist` [33] is an open source Python source separation library, but it is based on a different design framework than `nussl`, implements a different set of algorithms than `nussl`, and has no built-in interfaces for common evaluation metrics, data sets, or loading pre-trained models.

3. DESIGN FRAMEWORK OF `NUSSL`

`nussl` is built with extensibility in mind. It would be impossible to provide implementations for every source separation algorithm upon the announcement of this library. As such `nussl` is built to an API so that the community can easily add their own algorithms, models, datasets and have them automatically work with every other aspect of `nussl`.

Under the hood, `nussl` uses many common Python tools for signal processing and machine learning, such as `librosa`, `numpy`, `scipy`, `scikit-learn`, `mir_eval`, `musdb`, and `museval`, so developing with `nussl` should be familiar to any MIR researcher working in Python.

[2] https://interactiveaudiolab.github.io/nussl
[3] https://interactiveaudiolab.github.io/demos/nussl.html
[4] Related Python library: https://github.com/wslihgt/pyfasst/

```python
import nussl

# Load audio
signal = nussl.AudioSignal('path/to/mix.wav')

# Run REPET for foreground/background separation
algorithm = nussl.Repet(sig)
algorithm.run()
fg, bg = algorithm.make_audio_signals()

# Save results to wav files
fg.write_audio_to_file('fg.wav')
bg.write_audio_to_file('bg.wav')
```

Figure 1: Using `nussl` to run a single algorithm (REPET [30] for foreground/background separation) on a single mixture. In a recent seminar on reproducibility, REPET scored "could be reproduced, requiring extreme effort." `nussl` aims to improve the reproducibility score of multiple source separation algorithms, including REPET.

In the next sections, we provide a high-level overview of some of the more important aspects of the `nussl` API. For more information, please see our full online documentation.

3.1 `AudioSignal`

The main entry point to `nussl` for end-users and algorithm developers is through the `AudioSignal` object. The `AudioSignal` object has methods for reading and writing audio, padding or truncating the audio, adding and subtracting audio signals from one another, checking and altering properties of the audio, computing invertible signal transforms (e.g. short time Fourier transform), and much more. `AudioSignal` can read all of the most common audio codecs. Once in memory, audio is represented as a 2-dimensional (channels and time series within a channel) `numpy` array of pulse-code modulated (PCM) samples.

All source separation algorithms in `nussl` accept as their first argument an `AudioSignal` object. Each algorithm copies the content of the audio object, performs separation on that copy and returns a set of of new `AudioSignal` objects, one per source, leaving the original `AudioSignal` object unchanged.

3.2 Source Separation Algorithms

All source separation algorithms in `nussl` are encapsulated in classes that are derived from `SeparationBase`. For each class, the constructor does minimal set up, the `run()` method does the computation required for the source separation, and the `make_audio_signals()` method returns `AudioSignal` objects containing the estimated signals. An example of this whole process is shown in Figure 1.

3.2.1 `MaskSeparationBase` vs `SeparationBase`

Source separation algorithms in `nussl` are segregated into two categories: those that produce a mask and apply it

Algorithms in `nussl`			
Repetition	**Other Fore/Background**	**Spatialization**	**Composite**
Repet [31]	Harmonic/Percussive (HPSS) [7]	DUET [32]	Overlap/Add
RepetSim [29]	Melody Masking (Melodia [34])	PROJET [8]	Algorithm Picker [22]
2DFT [36]	**Component Analysis**	**Benchmarking**	**Neural Networks**
Matrix Decomposition	ICA [18]	High/Low Pass Filter	Deep Clustering [11, 21]
NMF w/ MFCC Clustering [38]	RPCA [12]	Ideal Mask	

Table 1: Source Separation algorithms by category currently implemented in `nussl`.

to a representation (e.g. a spectrogram) built from the waveform, and those that do separation via other means (e.g. time domain methods such as independent component analysis). The former group of algorithms are derived from the `MaskSeparationBase` base class, which is a subclass of the `SeparationBase` base class. The `run()` method in `MaskSeparationBase`-derived algorithms are expected to return mask objects (see Section 3.2.2). Some algorithms inherit directly from `SeparationBase` and have no requirement about what their `run()` method returns. With `MaskSeparationBase` separation classes, it is easy to switch between running an algorithm with a binary or soft mask.

3.2.2 Masks

Masks are encapsulated by the `MaskBase` base class. `SoftMask` and `BinaryMask` are the two classes that derive from `MaskBase`. `MaskBase`-derived objects have a `numpy` array that contains the data, and utilities for applying masks to `AudioSignal` objects. `SoftMask` objects are applied using a classical approach:

$$\hat{S}^{(i)}_{\omega,t} = \frac{v^{(i)}_{\omega,t}}{\sum_{i=0}^{N} v^{(i)}_{\omega,t}}$$

Here, $v^{(i)}_{\omega,t}$ is the estimate of source i at frequency ω and time t, $\hat{S}^{(i)}_{\omega,t}$ is the value of the mask for that source at that time and frequency, and N is the total number of sources. The `BinaryMask` objects simply put a 1 when a source estimate dominates all other source estimates in a time-frequency bin and a 0 elsewhere. More masking types (e.g. consistent Wiener filtering [17]) can be implemented by subclassing `MaskBase`.

3.3 Evaluation

`nussl` also has a common interface to evaluate the estimates from source separation algorithms using established metrics, such as BSS-Eval [6] using implementations from `mir_eval` [26] or `museval` [40]. `nussl` also has methods for comparing binary masks to an ideal binary mask using accuracy, precision, recall, and F-Score [43]. Similar to the rest of `nussl`, all of the evaluation metrics are encapsulated by the `EvaluationBase` base class so that all of its child classes are built to a common API.

```
import nussl

mir1k = 'path/to/MIR-1K'

# List of algorithms to test
sep_classes = [nussl.RepetSim, nussl.Melodia]

# Loop through all of MIR-1K
for mix, vox, acc in nussl.datasets.mir1k(mir1k):
    mix.to_mono(overwrite=True)

    for alg in sep_classes:

        # Run the algorithm
        a = alg(mix)
        a.run()
        est = a.make_audio_signals()

        # Evaluate results
        gt = [acc, vox] # Ground truth
        bss = nussl.evaluate.BssEval(mix, gt, est)
        scores = bss.evaluate()
```

Figure 2: Running two algorithms on *all* of MIR-1K and evaluating using BSS-Eval.

3.4 Data Sets

Although `nussl` does not ship with any data sets, it does provide "hooks" for interfacing with common data sets. The hooks are basic utilities for reading the audio files into `AudioSignal` objects. The user first points `nussl` to the top-level directory of the downloaded data set. The utilities can then iterate through every audio file, a subset of files, or shuffle the order in which they are read. There structure of the directories is assumed to be Data sets that `nussl` can currently interface with include iKala [2], MIR-1K [3], MUSDB18 [28] (using `musdb`), and DSD100 [20]. An example of running multiple algorithms on the entirety of MIR-1K and evaluating the results using BSS-Eval is shown in Figure 2.

3.5 Modelers and Deep Learning Models

`nussl` also contains a section for generic modeling and matrix manipulation classes. Classes in this section are not source separation algorithms, but are used by the algorithms in `nussl`. An example is the Non-negative Matrix Factorization (NMF) class, `NMF`. This receives a non-negative `numpy` matrix as input, factorizes it into a template matrix and an activation matrix, and outputs the two results to be used by a separation algorithm. It does not input or output audio, spectrograms, or masks. For those util-

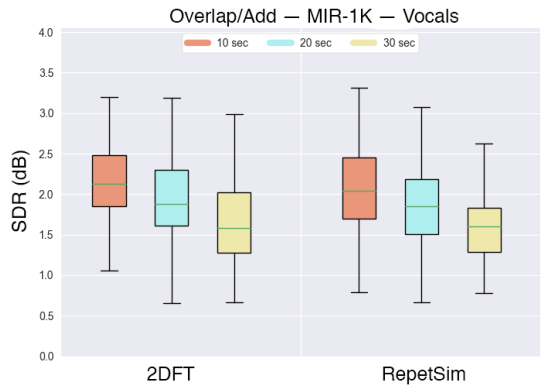


Figure 3: SDR evaluations for vocal estimates using `OverlapApp` on the MIR-1K data set. Three different window sizes are shown: 10 sec (red), 20 sec (blue), 30 sec (yellow). Hops are half of the window size.

ities, a wrapper separation class is needed, like `NMF_MFCC`, which clusters the templates using mel-frequency cepstral coefficients. Other classes train deep learning models for separation use. Classes in this section have a more lax API because of their heterogeneous nature.

`nussl` currently supports deep learning models written in `PyTorch`, but does not ship with any pre-trained models, only the code to train with. Similar to frameworks like `PyTorch`, `nussl` offers a way to download pre-trained models from `nussl` servers for algorithms which require them. Developers can see what models exist on our servers and download a models via utilities built into `nussl`. There also exists a process for contributers to upload their own pre-trained models (see Section 4.2 for more details).

4. ALGORITHMS IN `NUSSL`

4.1 Currently in `nussl`

At the time of this writing, the source separation algorithms are implemented in `nussl` to the API specification are presented in Table 1, by category. The algorithms currently in `nussl` provide a good starting point for future benchmark work, and we hope to expand the set of offered algorithms to include many more state-of-the-art approaches.

4.2 Adding new algorithms to `nussl`

The process of adding new source separation algorithms into nus`nussl`sl is similar to other open source projects, in many ways. A researcher who wishes to add an algorithm must clone the Github repository, make a new branch for their algorithm, add their code, push to Github, and then create a pull request. At this point, the `nussl` contributing process deviates from that standard open-source process.

After the new code passes the style and error checks, the researcher must provide benchmark files for tests. These can be created by using standard metrics on a set of example files. For example, when adding a new algorithm, a researcher could provide `BSS-Eval` metrics on a few songs from MIR-1K dataset. If an implementation existed elsewhere prior to being incorporated into `nussl`, then a copy of the original implementation will be requested to benchmark against. Authors of new algorithms, must also provide a reference to a paper or other documentation which outlines the algorithm in more detail. Additionally, any large supplemental materials that are needed for the algorithm (such as pre-trained neural network models) must be provided so that they can be distributed through `nussl`'s API as outlined in Section 3.5.

All of this is outlined in more detail on the contributions section of the `nussl` Github page and documentation.

5. EXAMPLE USES OF `NUSSL`

Because all of the algorithms and supporting infrastructure in `nussl` are built to an API, this allows a very simple way to find novel combinations of multiple source separation algorithms and evaluate them on a variety of data sets under different evaluation metrics. In this section, we will showcase two novel experiments using `nussl` and present results from these experiments.

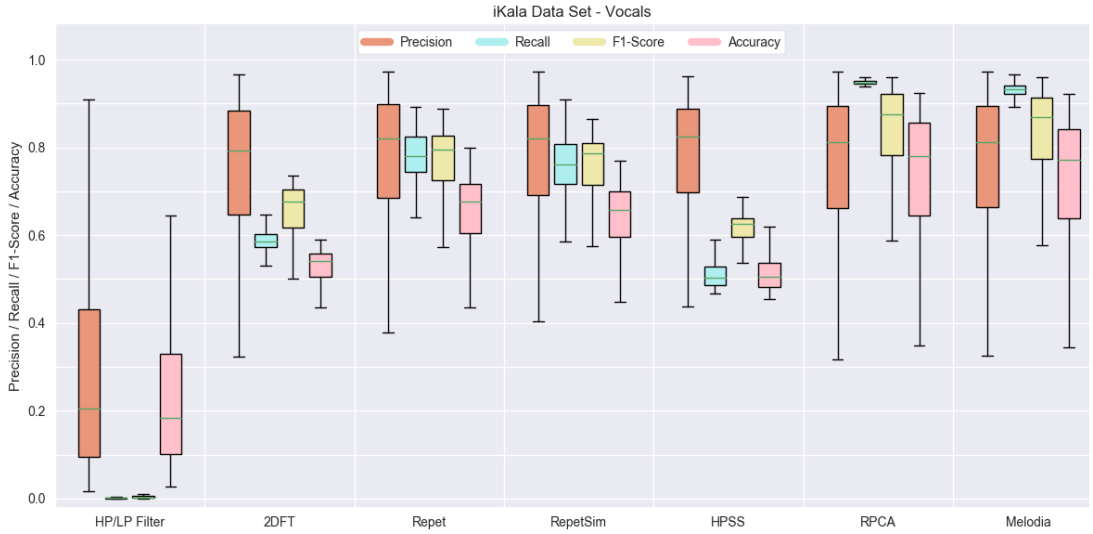
5.1 Cascading algorithms

The `nussl` API facilitates combining several different algorithms. To illustrate this point, we reproduce and expand upon work demonstrated by Rafii *et al.* [27] in combining rhythm-based and pitch-based approaches to source separation.

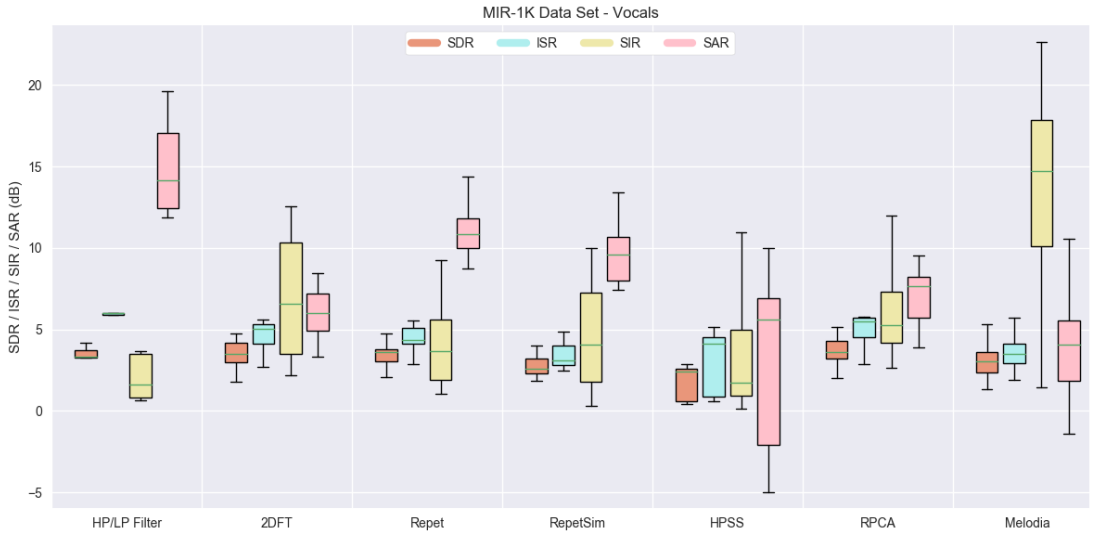
Rafii *et al.* present two methods for cascading algorithms: Parallel, where the background and foreground masks created by each algorithm are combined after the algorithms run on the mixture; and, Series, where the foreground estimation of algorithm A is fed in as the "mixture" to algorithm B. The mask estimates, in each case, are combined using weighted Weiner Filtering.

For this experiment, we use four background/foreground algorithms, RepetSim, Separation via 2DFT, RCPA, and Melodic masking with Melodia. We chose each pair from the set of algorithms and resulting in a total of 16 combinations. Based on values reported by Rafii *et al.*, for running in Parallel we set $w_B = 1.0$ and $w_M = 0.3$ as the weights of the background and foreground masks, respectively. We set the weight parameter $w = 0.5$ for running in Series. All algorithms created soft masks. We evaluated results using BSS-Eval on the undivided MIR-1K data set. [5] Mean SDR values (with 1 standard deviation) are shown in Figure 5 for vocals. We find that series configurations outperform parallel configurations overall, and RepetSim is best as a second algorithm run especially when it is also the first.

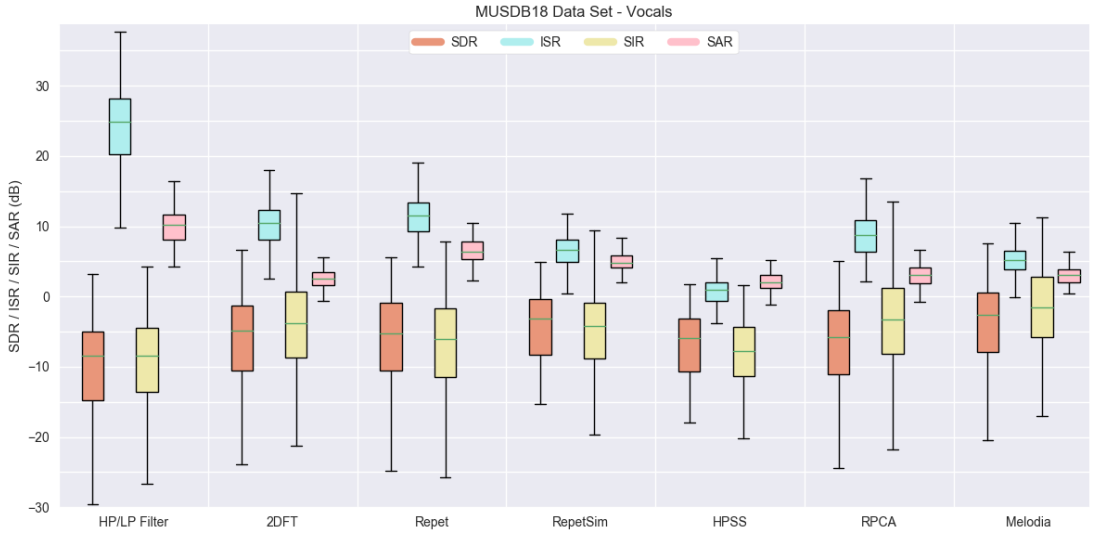
[5] MIR-1K has 110 tracks of mean duration 72.7 ± 17.3 seconds, that are divided into 1000 smaller tracks of 8.0 ± 1.8 seconds. The divided tracks are too small to capture multiple repetitions.



(a) Benchmarks for the iKala data set. Algorithms apply binary masks to the mixtures and the results were evaluated using precision, recall, F-Score, and accuracy (precision is red, recall is blue, F-Score is yellow, and accuracy is pink).

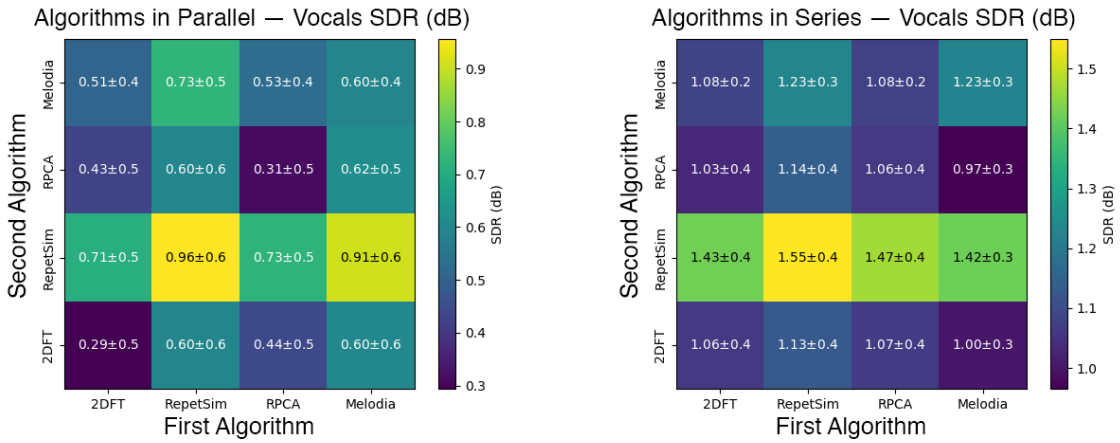


(b) Benchmarks for the MIR-1K data set. Algorithms apply binary masks to the mixtures and the results were evaluated using BSS-Eval (SDR is red, ISR is blue, SIR is yellow, and SAR is pink).



(c) Benchmarks for the MUSDB18 data set. Algorithms apply soft masks to the mixtures and the results were evaluated using BSS-Eval (SDR is red, ISR is blue, SIR is yellow, and SAR is pink).

Figure 4: Illustrative benchmarks for a set of algorithms and configurations in `nussl`.



(a) Running in algorithms in Parallel.

(b) Series

Figure 5: Mean SDR for vocal estimations from cascading pairs of algorithms.

5.2 Combining algorithms with Overlap/Add

In addition to classes that run one type of algorithm (like `HPSS`, etc), nussl also contains composite algorithm classes, *i.e.* those that run other algorithms in `nussl`. One such simple example is the `OverlapAdd` class, which does the overlap add method when running an algorithm.

In this experiment, we ran two repetition-based algorithms, RepetSim and Separation via 2DFT, wrapped in the Overlap/Add class to do vocal extraction. We tested three different window lengths, 10, 20, and 30 seconds, with hop length at half of the window length and using Hamming windows. We ran this experiment on the undivided MIR-1K data set [6] and evaluated the estimates using BSS-Eval. Results from this experiment show that smaller windows lead to better vocal separation performance, according to SDR. These results are shown in Figure 3.

6. BENCHMARKS

In this section, we provide a selection of benchmarks for a set of algorithms in `nussl`. We benchmarked all algorithms that explicitly perform vocal separation with deterministic output source ordering (*i.e.* for an output array of sources, accompaniment is always index 0 and vocals is always index 1). We ran the algorithms on the iKala, MIR-1K, and MUSDB18 data sets. We ran REPET, REPET-SIM, Separation by 2DFT, HPSS, Masking from Pitch Tracking (using Melodia as the pitch tracker), RPCA, High/Low Pass filtering (cutoff at 100Hz).

For brevity, we only report one evaluation type for each data set here. We aim not to be complete, but rather showcase what `nussl` is capable of. For iKala, we show precision/recall/F-Score/accuracy computed from output binary masks, Figure 4a. For MIR-1K, we show BSS-Eval metrics computed from estimates using binary masks, Figure 4b. And for MUSDB18, we show BSS-Eval metrics computed from estimates using soft masks, Figure 4c.

[6] We excluded two signals that were shorter than the largest window sizes.

All algorithms were run using the default parameter values for the algorithm in `nussl`'s implementation. Specifics of all of the parameters are contained in the project's documentation website.

7. FUTURE WORK AND CONCLUSION

In the future, we hope to expand upon `nussl` in a number of ways. First, while `nussl` is currently focused on musical source separation (the expertise of its authors), we would like to expand it to include source separation methods for speech. This would also necessitate adding hooks for speech data sets (like TIMIT and WSJ0) and adding pre-trained models for speech. Second, we would like to add an extensible API for spectral transformations. Currently, the STFT is at the core of `AudioSignal`, but in the future, it should be abstracted so that it is easy to run any algorithm on a CQT, Mel-Spaced STFT, etc.

Finally, and importantly, we would like buy-in from the MIR and audio community. The aim of `nussl` is to become the community's central repository for audio source separation. This goal is impossible without the support and contributions of the research community. We encourage interested participants to read the guidelines for contributing on this project's documentation page and get involved.

We have presented the Northwestern University Source Separation Library (`nussl`), an open-source, object-oriented audio source separation library implemented in Python. `nussl` implements many popular source separation algorithms, and a low barrier API for end-users and developers alike. We have demonstrated its design framework, including its ability to interface with common data sets and evaluation metrics. We also showcased two novel experiments using the API and a set of benchmarks. This project is actively seeking submissions from eager researchers and avid open source developers. Readers can find more information at `interactiveaudiolab.github.io/nussl`. This work was supported by USA National Science Foundation Award 1420971.

8. REFERENCES

[1] Rachel M. Bittner, Justin Salamon, Mike Tierney, Matthias Mauch, Chris Cannam, and Juan Pablo Bello. MedleyDB: A Multitrack Dataset for Annotation-Intensive MIR Research. *15th International Society for Music Information Retrieval Conference, ISMIR 2014, Taipei, Taiwan, October 27-31, 2014*, pages 155–160, 2014.

[2] Tak-Shing Chan, Tzu-Chun Yeh, Zhe-Cheng Fan, Hung-Wei Chen, Li Su, Yi-Hsuan Yang, and Roger Jang. Vocal Activity Informed Singing Voice Separation with the iKala Dataset. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 718–722, April 2015.

[3] Chao-Ling Hsu and J.-S.R. Jang. On the Improvement of Singing Voice Separation for Monaural Recordings Using the MIR-1K Dataset. *IEEE Transactions on Audio, Speech, and Language Processing*, 18(2):310–319, February 2010.

[4] Chris Drummond and Nathalie Japkowicz. Warning: Statistical Benchmarking is Addictive. Kicking the Habit in Machine Learning. *Journal of Experimental & Theoretical Artificial Intelligence*, 22(1):67–80, March 2010.

[5] Jean-Louis Durrieu, Bertrand David, and Gaël Richard. A Musically Motivated Mid-Level Representation for Pitch Estimation and Musical Audio Source Separation. *IEEE Journal of Selected Topics in Signal Processing*, 5(6):1180–1191, October 2011.

[6] Cédric Févotte, Rémi Gribonval, and Emmanuel Vincent. Bss_eval toolbox user guide–revision 2.0. page 19, 2005.

[7] Derry FitzGerald. Harmonic/Percussive Separation Using Median Filtering. *Proceedings of the 13th International Conference on Digital Audio Effects*, 2010.

[8] Derry FitzGerald, Antoine Liutkus, and Roland Badeau. PROJET — Spatial Audio Separation Using Projections. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 36–40, March 2016.

[9] John Garofalo, David Graff, Doug Paul, and David Pallett. Continous speech recognition (csr-i) wall street journal (wsj0) news, complete. *Linguistic Data Consortium, Philadelphia*, 1993.

[10] John S Garofolo, Lori F Lamel, William M Fisher, Jonathan G Fiscus, and David S Pallett. Darpa timit acoustic-phonetic continous speech corpus cd-rom. nist speech disc 1-1.1. *NASA STI/Recon technical report n*, 93, 1993.

[11] John R. Hershey, Zhuo Chen, Jonathan Le Roux, and Shinji Watanabe. Deep Clustering: Discriminative Embeddings for Segmentation and Separation. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 31–35. IEEE, March 2016.

[12] Po-Sen Huang, Scott Deeann Chen, Paris Smaragdis, and Mark Hasegawa-Johnson. Singing-Voice Separation from Monaural Recordings Using Robust Principal Component Analysis. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 57–60. IEEE, March 2012.

[13] Po-Sen Huang, Minje Kim, Mark Hasegawa-Johnson, and Paris Smaragdis. Deep Learning for Monaural Speech Separation. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1562–1566. IEEE, May 2014.

[14] Po-Sen Huang, Minje Kim, Mark Hasegawa-Johnson, and Paris Smaragdis. Singing-Voice Separation from Monaural Recordings using Deep Recurrent Neural Networks. *15th International Society for Music Information Retrieval Conference, ISMIR 2014, Taipei, Taiwan, October 27-31, 2014*, pages 477–482, 2014.

[15] Yusuf Isik, Jonathan Le Roux, Zhuo Chen, Shinji Watanabe, and John R Hershey. Single-Channel Multi-Speaker Separation Using Deep Clustering. *arXiv preprint arXiv:1607.02173*, 2016.

[16] Minje Kim and Paris Smaragdis. Bitwise Neural Networks. In *International Conference on Machine Learning (ICML) Workshop on Resource-Efficient Machine Learning*, Lille, France, Jul 2015.

[17] Jonathan Le Roux and Emmanuel Vincent. Consistent Wiener Filtering for Audio Source Separation. *IEEE Signal Processing Letters*, 20(3):217–220, March 2013.

[18] Te-Won Lee. *Independent Component Analysis: Theory and Applications*. Springer, New York; London, 2011. OCLC: 752483521.

[19] Antoine Liutkus, Fabian-Robert Stöter, Zafar Rafii, Daichi Kitamura, Bertrand Rivet, Nobutaka Ito, Nobutaka Ono, and Julie Fontecave. The 2016 SIgnal Separation Evaluation Campaign. In *Latent Variable Analysis and Signal Separation - 13th International Conference, LVA/ICA 2017, Grenoble, France, February 21-23, 2017, Proceedings*, pages 323–332, 2017.

[20] Antoine Liutkus, Fabian-Robert Stöter, Zafar Rafii, Daichi Kitamura, Bertrand Rivet, Nobutaka Ito, Nobutaka Ono, and Julie Fontecave. The 2016 Signal Separation Evaluation Campaign. In *International Conference on Latent Variable Analysis and Signal Separation*, pages 323–332. Springer, 2017.

[21] Yi Luo, Zhuo Chen, John R. Hershey, Jonathan Le Roux, and Nima Mesgarani. Deep Clustering and Conventional Networks for Music Separation:

Stronger Together. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 61–65. IEEE, March 2017.

[22] Ethan Manilow, Prem Seetharaman, Fatemeh Pishdadian, and Bryan Pardo. Predicting Algorithm Efficacy for Adaptive Multi-Cue Source Separation. In *2017 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, pages 274–278, Oct 2017.

[23] Alexey Ozerov and Emmanuel Vincent. Using the FASST source separation toolbox for noise robust speech recognition. In *International Workshop on Machine Listening in Multisource Environments (CHiME 2011)*, Florence, Italy, Sept 2011.

[24] Fatemeh Pishdadian, Bryan Pardo, and Antoine Liutkus. A Multi-resolution Approach to Common Fate-based Audio Separation. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 566–570. IEEE, March 2017.

[25] Po-Sen Huang, Minje Kim, Mark Hasegawa-Johnson, and Paris Smaragdis. Joint Optimization of Masks and Deep Recurrent Neural Networks for Monaural Source Separation. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 23(12):2136–2147, December 2015.

[26] Colin Raffel, Brian McFee, Eric J. Humphrey, Justin Salamon, Oriol Nieto, Dawen Liang, and Daniel P. W. Ellis. MIR_EVAL: A Transparent Implementation of Common MIR Metrics. In *15th International Society for Music Information Retrieval Conference, ISMIR 2014, Taipei, Taiwan, October 27-31, 2014*, pages 367–372, 2014.

[27] Zafar Rafii, Zhiyao Duan, and Bryan Pardo. Combining Rhythm-Based and Pitch-Based Methods for Background and Melody Separation. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 22(12):1884–1893, December 2014.

[28] Zafar Rafii, Antoine Liutkus, Fabian-Robert Stter, Stylianos Ioannis Mimilakis, and Rachel Bittner. The MUSDB18 corpus for music separation, December 2017.

[29] Zafar Rafii and Bryan Pardo. Music/Voice Separation Using the Similarity Matrix. *13th International Society for Music Information Retrieval Conference, ISMIR 2012, Mosteiro S.Bento Da Vitória, Porto, Portugal, October 8-12, 2012*, pages 583–588, 2012.

[30] Zafar Rafii and Bryan Pardo. REpeating Pattern Extraction Technique (REPET): A Simple Method for Music/Voice Separation. *IEEE Transactions on Audio, Speech, and Language Processing*, 21(1):73–84, Jan 2013.

[31] Zafar Rafii and Bryan Pardo. REpeating Pattern Extraction Technique (REPET): A Simple Method for Music/Voice Separation. *IEEE Transactions on Audio, Speech, and Language Processing*, 21(1):73–84, January 2013.

[32] Scott Rickard. The DUET blind source separation algorithm. In *Blind Speech Separation*, pages 217–241. Springer, 2007.

[33] Gerard Roma, Emad M Grais, Andrew JR Simpson, Iwona Sobieraj, and Mark D Plumbley. Untwist: A New Toolbox for Audio Source Separation. *Extended abstracts for the Late-Breaking Demo Session of the 17th International Society for Music Information Retrieval Conference, ISMIR 2016, New York City, United States, August 7-11, 2016*, 2016.

[34] Justin Salamon and Emilia Gomez. Melody Extraction From Polyphonic Music Signals Using Pitch Contour Characteristics. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(6):1759–1770, August 2012.

[35] Yann Salaün, Emmanuel Vincent, Nancy Bertin, Nathan Souviraa-Labastie, Xabier Jaureguiberry, Dung T Tran, and Frédéric Bimbot. The Flexible Audio Source Separation Toolbox Version 2.0. In *ICASSP*, 2014.

[36] Prem Seetharaman, Fatemeh Pishdadian, and Bryan Pardo. Music/Voice Separation Using the 2D Fourier Transform. In *2017 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, pages 36–40, Oct 2017.

[37] Paris Smaragdis and J.C. Brown. Non-Negative Matrix Factorization for Polyphonic Music Transcription. In *2003 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (IEEE Cat. No.03TH8684)*, pages 177–180. IEEE, 2003.

[38] Martin Spiertz and Volker Gnann. Source-Filter Based Clustering for Monaural Blind Source Separation. *Proceedings of the 12th International Conference on Digital Audio Effects*, 2009.

[39] Fabian-Robert Stoter, Antoine Liutkus, Roland Badeau, Bernd Edler, and Paul Magron. Common Fate Model for Unison Source Separation. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 126–130. IEEE, March 2016.

[40] Fabian-Robert Stöter, Antoine Liutkus, and Nobutaka Ito. The 2018 signal separation evaluation campaign. In *International Conference on Latent Variable Analysis and Signal Separation*, pages 293–305. Springer, 2018.

[41] Patrick Vandewalle, Jelena Kovacevic, and Martin Vetterli. Reproducible Research in Signal Processing.

IEEE Signal Processing Magazine, 26(3):37–47, May 2009.

[42] Emmanuel Vincent, Shoko Araki, Fabian Theis, Guido Nolte, Pau Bofill, Hiroshi Sawada, Alexey Ozerov, Vikrham Gowreesunker, Dominik Lutter, and Ngoc Q.K. Duong. The SIgnal Separation Evaluation Campaign (2007-2010): Achievements and Remaining Challenges. *Signal Processing*, 92(8):1928–1936, August 2012.

[43] DeLiang Wang. On ideal binary mask as the computational goal of auditory scene analysis. In *Speech separation by humans and machines*, pages 181–197. Springer, 2005.

[44] Guy Wolf, Stephane Mallat, and Shihab Shamma. Rigid Motion Model for Audio Source Separation. *IEEE Transactions on Signal Processing*, 64(7):1822–1831, April 2016.