mir_eval: A TRANSPARENT IMPLEMENTATION OF COMMON MIR METRICS

Colin Raffel[1,*], Brian McFee[1,2], Eric J. Humphrey[3], Justin Salamon[3,4], Oriol Nieto[3], Dawen Liang[1], and Daniel P. W. Ellis[1]

[1]LabROSA, Dept. of Electrical Engineering, Columbia University, New York
[2]Center for Jazz Studies, Columbia University, New York
[3]Music and Audio Research Lab, New York University, New York
[4]Center for Urban Science and Progress, New York University, New York

ABSTRACT

Central to the field of MIR research is the evaluation of algorithms used to extract information from music data. We present `mir_eval`, an open source software library which provides a transparent and easy-to-use implementation of the most common metrics used to measure the performance of MIR algorithms. In this paper, we enumerate the metrics implemented by `mir_eval` and quantitatively compare each to existing implementations. When the scores reported by `mir_eval` differ substantially from the reference, we detail the differences in implementation. We also provide a brief overview of `mir_eval`'s architecture, design, and intended use.

1. EVALUATING MIR ALGORITHMS

Much of the research in Music Information Retrieval (MIR) involves the development of systems that process raw music data to produce semantic information. The goal of these systems is frequently defined as attempting to duplicate the performance of a human listener given the same task [5]. A natural way to determine a system's effectiveness might be for a human to study the output produced by the system and judge its correctness. However, this would yield only subjective ratings, and would also be extremely time-consuming when evaluating a system's output over a large corpus of music.

Instead, objective metrics are developed to provide a well-defined way of computing a score which indicates each system's output's correctness. These metrics typically involve a heuristically-motivated comparison of the system's output to a reference which is known to be correct. Over time, certain metrics have become standard for each task, so that the performance of systems created by different researchers can be compared when they are evaluated over the same dataset [5]. Unfortunately, this comparison can be confounded by small details of the implementations or procedures that can have disproportionate impacts on the resulting scores.

For the past 10 years, the yearly Music Information Retrieval Evaluation eXchange (MIREX) has been a forum for comparing MIR algorithms over common datasets [6]. By providing a standardized shared-task setting, MIREX has become critically useful for tracking progress in MIR research. MIREX is built upon the Networked Environment for Music Analysis (NEMA) [22], a large-scale system which includes exhaustive functionality for evaluating, summarizing, and displaying evaluation results. The NEMA codebase includes multiple programming languages and dependencies (some of which, *e.g.* Matlab, are proprietary) so compiling and running it at individual sites is nontrivial. In consequence, the NEMA system is rarely used for evaluating MIR algorithms outside of the setting of MIREX [6]. Instead, researchers often create their own implementations of common metrics for evaluating their algorithms. These implementations are thus not standardized, and may contain differences in details, or even bugs, that confound comparisons.

These factors motivate the development of a standardized software package which implements the most common metrics used to evaluate MIR systems. Such a package should be straightforward to use and well-documented so that it can be easily adopted by MIR researchers. In addition, it should be community-developed and transparently implemented so that all design decisions are easily understood and open to discussion and improvement.

Following these criteria, we present `mir_eval`, a software package which intends to provide an easy and standardized way to evaluate MIR systems. This paper first discusses the architecture and design of `mir_eval` in Section 2, then, in Section 3, describes all of the tasks covered by `mir_eval` and the metrics included. In order to validate our implementation decisions, we compare `mir_eval` to existing software in Section 4. Finally, we discuss and summarize our contributions in Section 5.

*Please direct correspondence to craffel@gmail.com

2. mir_eval'S ARCHITECTURE

mir_eval is a Python library which currently includes metrics for the following tasks: Beat detection, chord estimation, pattern discovery, structural segmentation, melody extraction, and onset detection. Each task is given its own submodule, and each metric is defined as a separate function in each submodule. Each task submodule also includes common data pre-processing steps for the task. Every metric function includes detailed documentation, example usage, input validation, and references to the original paper which defined the metric. mir_eval also includes a submodule io which provides convenience functions for loading in task-specific data from common file formats (*e.g.* comma/tab separated values, .lab files [7], *etc.*). For readability, all code follows the PEP8 style guide [21]. mir_eval's only dependencies outside of the Python standard library are the free and open-source SciPy/Numpy [9] and scikit-learn [15] libraries.

In order to simplify the usage of mir_eval, it is packaged with a set of "evaluator" scripts, one for each task. These scripts include all code necessary to load in data, pre-process it, and compute all metrics for a given task. The evaluators allow for mir_eval to be called directly from the command line so that no knowledge of Python is necessary. They are also distributed as executables for Windows and Mac OS X, so that mir_eval may be used with no dependencies installed.

3. TASKS INCLUDED IN mir_eval

In this section, we enumerate the tasks and metrics implemented in mir_eval. Due to space constraints, we only give high-level descriptions for each metric; for exact definitions see the references provided.

3.1 Beat Detection

The aim of a beat detection algorithm is to report the times at which a typical human listener might tap their foot to a piece of music. As a result, most metrics for evaluating the performance of beat tracking systems involve computing the error between the estimated beat times and some reference list of beat locations. Many metrics additionally compare the beat sequences at different metric levels in order to deal with the ambiguity of tempo [4].

mir_eval includes the following metrics for beat tracking, which are defined in detail in [4]: The **F-measure** of the beat sequence, where an estimated beat is considered correct if it is sufficiently close to a reference beat; **Cemgil's score**, which computes the sum of Gaussian errors for each beat; **Goto's score**, a binary score which is 1 when at least 25% of the estimated beat sequence closely matches the reference beat sequence; **McKinney's P-score**, which computes the cross-correlation of the estimated and reference beat sequences represented as impulse trains; **continuity-based scores** which compute the proportion of the beat sequence which is continuously correct; and finally the **Information Gain** of a normalized beat error histogram over a uniform distribution.

3.2 Chord Estimation

Despite being one of the oldest MIREX tasks, evaluation methodology and metrics for automatic chord estimation is an ongoing topic of discussion, due to issues with vocabularies, comparison semantics, and other lexicographical challenges unique to the task [14]. One source of difficulty stems from an inherent subjectivity in "spelling" a chord name and the level of detail a human observer can provide in a reference annotation [12]. As a result, a consensus has yet to be reached regarding the single best approach to comparing two sequences of chord labels, and instead are often compared over a set of rules, i.e Root, Major-Minor, and Sevenths, with or without inversions.

To efficiently compare chords, we first separate a given chord label into a its constituent parts, based on the syntax of [7]. For example, the chord label G:maj(6)/5 is mapped to three pieces of information: the root ("G"), the root-invariant active semitones as determined by the quality shorthand ("maj") and scale degrees ("6"), and the bass interval ("5").

Based on this representation, we can compare an estimated chord label with a reference by the following rules as used in MIREX 2013 [2]: **Root** requires only that the roots are equivalent; **Major-Minor** includes Root, and further requires that the active semitones are equivalent subject to the reference chord quality being Maj or min; **Sevenths** follows Major-minor, but is instead subject to the reference chord quality being one of Maj, min, Maj7, min7, 7, or minmaj7; and finally, **Major-Minor-Inv** and **Sevenths-Inv** include Major-Minor and Sevenths respectively, but further require that the bass intervals are equivalent subject to the reference bass interval being an active semitone. The "subject to..." conditions above indicate that a comparison is *ignored* during evaluation if the given criteria is not satisfied.

Track-wise scores are computed by weighting each comparison by the duration of its interval, over all intervals in an audio file. This is achieved by forming the union of the boundaries in each sequence, sampling the labels, and summing the time intervals of the "correct" ranges. The cumulative score, referred to as *weighted chord symbol recall*, is tallied over a set audio files by discrete summation, where the importance of each score is weighted by the duration of each annotation [2].

3.3 Pattern Discovery

Pattern discovery involves the identification of musical patterns (*i.e.* short fragments or melodic ideas that repeat at least twice) both from audio and symbolic representations. The metrics used to evaluation pattern discovery systems attempt to quantify the ability of the algorithm to not only determine the present patterns in a piece, but also to find all of their occurrences.

Collins compiled all previously existent metrics and proposed novel ones [3] which resulted in 19 different scores, each one implemented in mir_eval: **Standard F-measure, Precision, and Recall**, where an estimated prototype pattern is considered correct only if it matches

(up to translation) a reference prototype pattern; **Establishment F-measure, Precision, and Recall**, which compute the number of reference patterns that were successfully found, no matter how many occurrences were found; **Occurrence F-measure, Precision, and Recall**, which measure whether an algorithm is able to retrieve all occurrences of a pattern; **Three-layer F-measure, Precision, and Recall**, which capture both the establishment of the patterns and the occurrence retrieval in a single set of scores; and the **First N patterns metrics**, which compute the target proportion establishment recall and three-layer precision for the first N patterns only in order to measure the ability of the algorithm to sort the identified patterns based on their relevance.

3.4 Structural Segmentation

Evaluation criteria for structural segmentation fall into two categories: boundary annotation and structural annotation. Boundary annotation is the task of predicting the times at which structural changes occur, such as when a verse transitions to a refrain. Structural annotation is the task of assigning labels to detected segments. The estimated labels may be arbitrary strings — such as A, B, C, *etc.* — and they need not describe functional concepts. In both tasks, we assume that annotations express a partitioning of the track into intervals.

`mir_eval` implements the following boundary detection metrics: **Boundary Detection Precision, Recall, and F-measure Scores** where an estimated boundary is considered correct if it falls within a window around a reference boundary [20]; and **Boundary Deviation** which computes median absolute time difference from a reference boundary to its nearest estimated boundary, and vice versa [20]. The following structure annotation metrics are also included: **Pairwise Classification Precision, Recall, and F-measure Scores** for classifying pairs of sampled time instants as belonging to the same structural component [10]; **Rand Index** [1] which clusters reference and estimated annotations and compares them by the Rand Index [17]; and the **Normalized Conditional Entropy** where sampled reference and estimated labels are interpreted as samples of random variables Y_R, Y_E from which the conditional entropy of Y_R given Y_E (**Under-Segmentation**) and Y_E given Y_R (**Over-Segmentation**) are estimated [11].

3.5 Melody Extraction

Melody extraction algorithms aim to produce a sequence of frequency values corresponding to the pitch of the dominant melody from a musical recording [19]. An estimated pitch series is evaluated against a reference by computing the following five measures defined in [19], first used in MIREX 2005 [16]: **Voicing Recall Rate** which computes the proportion of frames labeled as melody frames in the reference that are estimated as melody frames by the algorithm; **Voicing False Alarm Rate** which computes the proportion of frames labeled as non-melody in the reference that are mistakenly estimated as melody frames by the algorithm; **Raw Pitch Accuracy** which computes the proportion of melody frames in the reference for which the frequency is considered correct (*i.e.* within half a semitone of the reference frequency); **Raw Chroma Accuracy** where the estimated and reference f_0 sequences are mapped onto a single octave before computing the raw pitch accuracy; and the **Overall Accuracy**, which computes the proportion of all frames correctly estimated by the algorithm, including whether non-melody frames where labeled by the algorithm as non-melody. Prior to computing these metrics, both the estimate and reference sequences must be sampled onto the same time base.

[1] The MIREX results page refers to Rand Index as "random clustering index".

3.6 Onset Detection

The goal of an onset detection algorithm is to automatically determine when notes are played in a piece of music. As is also done in beat tracking and segment boundary detection, the primary method used to evaluate onset detectors is to first determine which estimated onsets are "correct", where correctness is defined as being within a small window of a reference onset [1]. From this, **Precision**, **Recall**, and **F-measure** scores are computed.

4. COMPARISON TO EXISTING IMPLEMENTATIONS

In order to validate the design choices made in `mir_eval`, it is useful to compare the scores it reports to those reported by an existing evaluation system. Beyond pinpointing intentional differences in implementation, this process can also help find and fix bugs in either `mir_eval` or the system it is being compared to.

For each task covered by `mir_eval`, we obtained a collection of reference and estimated annotations and computed or obtained a score for each metric using `mir_eval` and the evaluation system being compared to. In order to facilitate comparison, we ensured that all parameters and pre-processing used by `mir_eval` were equivalent to the reference system unless otherwise explicitly noted. Then, for each reported score, we computed the relative change between the scores as their absolute difference divided by their mean, or

$$\frac{|s_m - s_c|}{(s_m + s_c)/2}$$

where s_m is the score reported by `mir_eval` and s_c is the score being compared to. Finally, we computed the average relative change across all examples in the obtained dataset for each score.

For the beat detection, chord estimation, structural segmentation, and onset detection tasks, MIREX releases the the output of submitted algorithms, the ground truth annotations, and the reported score for each example in each data set. We therefore can directly compare `mir_eval` to MIREX for these tasks by collecting all reference and estimated annotations, computing each metric for each example using identical pre-processing and parameters as appropriate, and comparing the result to the score reported by

MIREX. We chose to compare against the results reported in MIREX 2013 for all tasks.

In contrast to the tasks listed above, MIREX does not release ground truth annotations or algorithm output for the melody extraction and pattern discovery tasks. As a result, we compared `mir_eval`'s output on smaller development datasets for these tasks. For melody extraction, the ADC2004 dataset used by MIREX is publicly available. We performed melody extraction using the "SG2" algorithm evaluated in 2011 [18] and compared `mir_eval`'s reported scores to those of MIREX. For pattern discovery, we used the development dataset released by Collins [3] and used the algorithms submitted by Nieto and Farbood [13] for MIREX 2013 to produce estimated patterns. We evaluated the estimated patterns using the MATLAB code released by Collins [3]. The number of algorithms, examples, and total number of scores for all tasks are summarized in Table 1.

Task	Algorithms	Examples	Scores
Beat Detection	20	679	13580
Segmentation	8	1397	11176
Onset Detection	11	85	935
Chord Estimation	12	217	2604
Melody	1	20	20
Pattern Discovery	4	5	20

Table 1. Number of scores collected for each task for comparison against `mir_eval`.

The resulting average relative change for each metric is presented in Table 2. The average relative change for all of the pattern discovery metrics was 0, so they are not included in this table. For many of the other metrics, the average relative change was less than a few tenths of a percent, indicating that `mir_eval` is equivalent up to rounding/precision errors. In the following sections, we enumerate the known implementation differences which account for the larger average relative changes.

4.1 Non-greedy matching of events

In the computation of the F-measure, Precision and Recall metrics for the beat tracking, boundary detection, and onset detection tasks, an estimated event is considered correct (a "hit") if it falls within a small window of a reference event. No estimated event is counted as a hit for more than one reference event, and vice versa. In MIREX, this assignment is done in a greedy fashion, however in `mir_eval` we use an optimal matching strategy. This is accomplished by computing a maximum bipartite matching between the estimated events and the reference events (subject to the window constraint) using the Hopcroft-Karp algorithm [8]. This explains the observed discrepancy between `mir_eval` and MIREX for each of these metrics. In all cases where the metric differs, `mir_eval` reports a higher score, indicating that the greedy matching strategy was suboptimal.

4.2 McKinney's P-score

When computing McKinney's P-score [4], the beat sequences are first converted to impulse trains sampled at a 10 millisecond resolution. Because this sampling involves quantizing the beat times, shifting both beat sequences by a constant offset can result in substantial changes in the P-score. As a result, in `mir_eval`, we normalize the beat sequences by subtracting from each reference and estimated beat location the minimum beat location in either series. In this way, the smallest beat in the estimated and reference beat sequences is always 0 and the metric remains the same even when both beat sequences have a constant offset applied. This is not done in MIREX (which uses the Beat Evaluation Toolbox [4]), and as a result, we observe a considerable average relative change for the P-score metric.

4.3 Information Gain

The Information Gain metric [4] involves the computation of a histogram of the per-beat errors. The Beat Evaluation Toolbox (and therefore MIREX) uses a non-uniform histogram binning where the first, second and last bins are smaller than the rest of the bins while `mir_eval` uses a standard uniformly-binned histogram. As a result, the Information Gain score reported by `mir_eval` differs substantially from that reported by MIREX.

4.4 Segment Boundary Deviation

When computing the median of the absolute time differences for the boundary deviation metrics, there are often an even number of reference or estimated segment boundaries, resulting in an even number of differences to compute the median over. In this case, there is no "middle" element to choose as the median. `mir_eval` follows the typical convention of computing the mean of the two middle elements in lieu of the median for even-length sequences, while MIREX chooses the larger of the two middle elements. This accounts for the discrepancy in the reference-to-estimated and estimated-to-reference boundary deviation metrics.

4.5 Interval Sampling for Structure Metrics

When computing the structure annotation metrics (Pairwise Precision, Recall, and F-measure, Rand Index, and Normalized Conditional Entropy Over- and Under-Segmentation Scores), the reference and estimated labels must be sampled to a common time base. In MIREX, a fixed sampling grid is used for the Rand Index and pairwise classification metrics, but a different sampling rate is used for each, while a fixed number of samples is used for the conditional entropy scores. In `mir_eval`, the same fixed sampling rate of 100 milliseconds is used for all structure annotation metrics, as specified in [23].

Furthermore, in MIREX the start and end time over which the intervals are sampled depends on both the reference and estimated intervals while `mir_eval` always samples with respect to the reference to ensure fair comparison across multiple estimates. This additionally requires

Beat Detection								
F-measure	Cemgil	Goto	P-score	CMLc	CMLt	AMLc	AMLt	In. Gain
0.703%	0.035%	0.054%	0.877%	0.161%	0.143%	0.137%	0.139%	9.174%

Structural Segmentation								
NCE-Over	NCE-under	Pairwise F	Pairwise P	Pairwise R	Rand	F@.5	P@.5	R@.5
3.182%	11.082%	0.937%	0.942%	0.785%	0.291%	0.429%	0.088%	1.021%

Structural Segmentation (continued)					Onset Detection		
F@3	P@3	R@3	Ref-est dev.	Est-ref dev.	F-measure	Precision	Recall
0.393%	0.094%	0.954%	0.935%	0.000%	0.165%	0.165%	0.165%

Chord Estimation					Melody Extraction				
Root	Maj/min	Maj/min + Inv	7ths	7ths + Inv	Overall	Raw pitch	Chroma	Voicing R	Voicing FA
0.007%	0.163%	1.005%	0.483%	0.899%	0.070%	0.087%	0.114%	0.000%	10.095%

Table 2. Average relative change for every metric in `mir_eval` when compared to a pre-existing implementation. The average relative change for all pattern discovery metrics was 0, so they are not shown here.

that estimated intervals are adjusted to span the exact duration specified by the reference intervals. This is done by adding synthetic intervals when the estimated intervals do not span the reference intervals or otherwise trimming estimated intervals. These differences account for the average relative changes for the structure annotation metrics.

4.6 Segment Normalized Conditional Entropy

When adding intervals to the estimated annotation as described above, `mir_eval` ensures that the labels do not conflict with existing labels. This has the effect of changing the normalization constant in the Normalized Conditional Entropy scores. Furthermore, when there's only one label, the Normalized Conditional Entropy scores are not well defined. MIREX assigns a score of 1 in this case; `mir_eval` assigns a score of 0. This results in a larger average change for these two metrics.

4.7 Melody Voicing False Alarm Rate

When a reference melody annotation contains no unvoiced frames, the Voicing False Alarm Rate is not well defined. MIREX assigns a score of 1 in this case, while `mir_eval` assigns a score of 0 because, intuitively, no reference unvoiced frames could be estimated, so no false alarms should be reported. In the data set over which the average relative change for the melody metrics was computed, one reference annotation contained no unvoiced frames. This discrepancy caused a large inflation of the average relative change reported for the Voicing False Alarm Rate due to the small number of examples in our dataset.

4.8 Weighted Chord Symbol Recall

The non-negligible average relative changes seen in the chord metrics are caused by two main sources of ambiguity. First, we find some chord labels in the MIREX reference annotations lack well-defined, *i.e.* singular, mappings into a comparison space. One such example is `D:maj(*1)/#1`. While the quality shorthand indicates "major", the asterisk implies the root is omitted and thus it is unclear whether `D:maj(*1)/#1` is equivalent to `D:maj1`. Second, and more importantly, such chords are likely ignored during evaluation, and we are unable to replicate the exact exclusion logic used by MIREX. This has proven to be particularly difficult in the two inversion rules, and manifests in Table 2. For example, `Bb:maj(9)/9` was *not* excluded from the MIREX evaluation, contradicting the description provided by the task specification [2]. This chord alone causes an observable difference between `mir_eval` and MIREX's results.

5. TOWARDS TRANSPARENCY AND COMMUNITY INVOLVEMENT

The results in Section 4 clearly demonstrate that differences in implementation can lead to substantial differences in reported scores. This corroborates the need for transparency and community involvement in comparative evaluation. The primary motivation behind developing `mir_eval` is to establish an open-source, publicly refined implementation of the most common MIR metrics. By encouraging MIR researchers to use the same easily understandable evaluation codebase, we can ensure that different systems are being compared fairly.

While we have given thorough consideration to the design choices made in `mir_eval`, we recognize that standards change over time, new metrics are proposed each year, and that only a subset of MIR tasks are currently implemented in `mir_eval`. Towards this end, `mir_eval` is hosted on Github, [2] which provides a straightforward way of proposing changes and additions to the codebase using the Pull Request feature. With active community participation, we believe that `mir_eval` can ensure that MIR research converges on a standard methodology for evaluation.

[2] `http://github.com/craffel/mir_eval`

6. ACKNOWLEDGEMENTS

The authors would like to thank Matthew McVicar for helpful advice on comparing chord labels and Tom Collins for sharing his MATLAB implementation to evaluate musical patterns. Support provided in part by The Andrew W. Mellon Foundation and the National Science Foundation, under grants IIS-0844654 and IIS-1117015.

7. REFERENCES

[1] S. Böck, F. Krebs, and M. Schedl. Evaluating the online capabilities of onset detection methods. In *Proceedings of the 13th International Society for Music Information Retrieval Conference (ISMIR)*, pages 49–54, 2012.

[2] K. Choi and J. A. Burgoyne. MIREX task: Audio chord estimation. `http://www.music-ir.org/mirex/wiki/2013:Audio_Chord_Estimation`, 2013. Accessed: 2014-04-30.

[3] T. Collins. MIREX task: Discovery of repeated themes & sections. `http://www.music-ir.org/mirex/wiki/2013:Discovery_of_Repeated_Themes_&_Sections`, 2013. Accessed: 2014-04-30.

[4] M. E. P. Davies, N. Degara, and M. D. Plumbley. Evaluation methods for musical audio beat tracking algorithms. Technical Report C4DM-TR-09-06, Centre for Digital Music, Queen Mary University of London, London, England, October 2009.

[5] J. S. Downie. Toward the scientific evaluation of music information retrieval systems. In *Proceedings of the 4th International Society for Music Information Retrieval Conference (ISMIR)*, pages 25–32, 2003.

[6] J. S. Downie. The music information retrieval evaluation exchange (2005-2007): A window into music information retrieval research. *Acoustical Science and Technology*, 29(4):247–255, 2008.

[7] C. Harte. *Towards Automatic Extraction of Harmony Information from Music Signals*. PhD thesis, Queen Mary University of London, August 2010.

[8] J. E. Hopcroft and R. M. Karp. An n^5/2 algorithm for maximum matchings in bipartite graphs. *SIAM Journal on computing*, 2(4):225–231, 1973.

[9] E. Jones, T. Oliphant, P. Peterson, et al. SciPy: Open source scientific tools for Python, 2001–.

[10] M. Levy and M. Sandler. Structural segmentation of musical audio by constrained clustering. *IEEE Transactions on Audio, Speech, and Language Processing*, 16(2):318–326, 2008.

[11] H. M. Lukashevich. Towards quantitative measures of evaluating song segmentation. In *Proceedings of the 9th International Society for Music Information Retrieval Conference (ISMIR)*, pages 375–380, 2008.

[12] Y. Ni, M. McVicar, R. Santos-Rodriguez, and T. De Bie. Understanding effects of subjectivity in measuring chord estimation accuracy. *IEEE Transactions on Audio, Speech, and Language Processing*, 21(12):2607–2615, 2013.

[13] O. Nieto and M. Farbood. Discovering musical patterns using audio structural segmentation techniques. *7th Music Information Retrieval Evaluation eXchange (MIREX)*, 2013.

[14] J. Pauwels and G. Peeters. Evaluating automatically estimated chord sequences. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 749–753. IEEE, 2013.

[15] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[16] G. E. Poliner, D. P. W. Ellis, A. F. Ehmann, E. Gómez, S. Streich, and B. Ong. Melody transcription from music audio: Approaches and evaluation. *IEEE Transactions on Audio, Speech, and Language Processing*, 15(4):1247–1256, 2007.

[17] W. M. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association*, 66(336):846–850, 1971.

[18] J. Salamon and E. Gómez. Melody extraction from polyphonic music signals using pitch contour characteristics. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(6):1759–1770, Aug. 2012.

[19] J. Salamon, E. Gómez, D. P. W. Ellis, and G. Richard. Melody extraction from polyphonic music signals: Approaches, applications and challenges. *IEEE Signal Processing Magazine*, 31(2):118–134, March 2014.

[20] D. Turnbull, G. Lanckriet, E. Pampalk, and M. Goto. A supervised approach for detecting boundaries in music using difference features and boosting. In *Proceedings of the 8th International Society for Music Information Retrieval Conference (ISMIR)*, pages 51–54, 2007.

[21] G. van Rossum, B. Warsaw, and N. Coghlan. PEP 8–style guide for python code. `http://www.python.org/dev/peps/pep-0008`, 2001. Accessed: 2014-04-30.

[22] K. West, A. Kumar, A. Shirk, G. Zhu, J. S. Downie, A. Ehmann, and M. Bay. The networked environment for music analysis (nema). In *IEEE 6th World Congress on Services (SERVICES 2010)*, pages 314–317. IEEE, 2010.

[23] C. Willis. MIREX task: Structural segmentation. `http://www.music-ir.org/mirex/wiki/2013:Structural_Segmentation`, 2013. Accessed: 2014-04-30.