mir_eval

First author
Affiliation1
author1@ismir.edu

Second author
Retain these fake authors in submission to preserve the formatting

Third author
Affiliation3
author3@ismir.edu

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 [4]. 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 [4]. 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 [5]. By providing a standardized shared-task setting, MIREX has become critically useful to track progress in MIR research. MIREX is built upon the Networked Environment for Music Analysis (NEMA) [18], 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 [5]. 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.

2. mir_eval'S ARCHITECTURE

mir_eval is a Python library which includes metrics for the following tasks: Beat detection, chord recognition, pattern discovery, structural segmentation, melody extraction, and onset detection. Each task is given its own submodule (e.g. mir_eval.beat), and each metric is defined as a separate function in each submodule (e.g. mir_eval.beat.f_measure). 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. For readability, all code follows the PEP8 style guide [17]. `mir_eval`'s only dependencies outside of the Python standard library are the free and open-source SciPy/Numpy [6] and scikit-learn [11] Python 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`

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 [7].

`mir_eval` includes the following metrics for beat tracking, which are defined in detail in [3]: 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 some specific heuristic criteria are met; **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 the beat error histogram to a uniform distribution.

3.2 Chord Estimation

Despite being one of the oldest MIREX tasks, evaluation methodology and metrics for automatic chord estimation (ACE) is an ongoing topic of discussion, due to issues with vocabularies, comparison semantics, and other lexicographical challenges unique to the task [**?**]. 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 [**?**]. 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 translate a given chord label $\mathcal{C}$ into a numerical representation, based on the syntax of [**?**]. For example, $G : maj(6)/5$ is mapped to three pieces of information: one, the root ("G") is mapped to an absolute pitch class $\mathcal{R}$, in $[0, 11]$, where $C \rightarrow 0$, $C\sharp/D\flat \rightarrow 1$, etc; two, the quality shorthand ("maj") and scale degrees ("6") are mapped to a root-invariant bit vector $\mathcal{S}$ of active semitones; and three, the bass interval ("5") is translated to the relative scale degree in semitones $\mathcal{B}$.

Based on this representation, we can symbolically define our interpretation of the five rules used in MIREX 2013 [1] to compare an estimated chord $\mathcal{C}_{est}$ with a reference $\mathcal{C}_{ref}$. **Root** is given by $\mathcal{R}_{est} == \mathcal{R}_{ref}$, for all chords. **Major-Minor** includes **Root**, plus $\mathcal{S}_{est} == \mathcal{S}_{ref}$ subject to $\mathcal{S}_{ref} \in \{Maj, min\}$. **Major-Minor-Inv** includes **Major-minor**, plus $\mathcal{B}_{est} == \mathcal{B}_{ref}$ subject to $\mathcal{B}_{ref} \in \mathcal{S}_{ref}$. **Sevenths** follows **Major-minor**, but is instead subject to $\mathcal{S}_{ref} \in \{Maj, min, Maj7, min7, 7, minmaj7\}$. Finally, **Sevenths-Inv** includes **Sevenths**, plus $\mathcal{B}_{est} == \mathcal{B}_{ref}$ subject to $\mathcal{B}_{ref} \in \mathcal{S}_{ref}$. Before proceeding, we draw attention to the set-inclusion conditions above, e.g. "subject to...", meaning that a comparison is *ignored* during evaluation if the given criteria is not satisfied.

Track-wise scores, S_n, are computed by weighting each comparison by the duration of its interval, over all intervals in an audio file; stated another way, this is the piecewise continuous-time integral of the equivalence of two chord sequences, $(\mathbf{C}_{ref}, \mathbf{C}_{est})$, expressed as follows:

$$S_n(\mathbf{C}_{ref}, \mathbf{C}_{est}) = \frac{1}{T} \int_{t=0}^{T} \mathcal{C}_{ref}(t) == \mathcal{C}_{est}(t) \qquad (1)$$

This is achieved by forming the union of the boundaries in each sequence, and summing the time intervals of the "correct" ranges. The cumulative score, referred to as *weighted chord symbol recall* (WCSR), is tallied over a set of N items, i.e. audio files, by discrete summation, where the importance of each score, S_n, is weighted by the duration, T_n, of each annotation:

$$WCSR_{total} = \frac{\sum_{n=0}^{N} T_n * S_n}{\sum_{n=0}^{N} T_n} \qquad (2)$$

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 presents pattern in a piece, but also to find all of their occurrences. Given the novelty of this task (the first year it appeared in MIREX was 2013), the evaluation metrics are likely to be modified in further editions. Nonetheless, Collins compiled all previously existent metrics and proposed novel ones [2], which resulted in 19 different scores, each one implemented in `mir_eval` and described below:

Standard F-measure, Precision, and Recall: An estimated pattern is considered correct only if it matches (up to translation) a reference prototype pattern.

[1] http://www.music-ir.org/mirex/wiki/2013:Audio_Chord_Estimation

Establishment F-measure, Precision, and Recall: Computes the number of reference patterns that were successfully found, no matter how many occurrences were found. In other words, these metrics capture the ability of the algorithm to *establish* the patterns in a piece.

Occurrence F-measure, Precision, and Recall: Measures whether an algorithm is able to retrieve all occurrences of a pattern. This metric aims to quantize this *occurrence* retrieval quality of the algorithm.

Three-layer F-measure, Precision, and Recall: These metrics capture both the establishment of the patterns and the occurrence retrieval in a single set of scores.

First N patterns metrics: The target proportion establishment recall and three-layer precision analyzing the first N patterns only, which measures 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 [16];

Boundary deviation: median absolute time difference from a reference boundary to its nearest estimated boundary, and vice versa [16],

and the following structure annotation metrics:

Pairwise classification: precision, recall, and F-measure for classifying pairs of sampled time instants as belonging to the same structural component [8];

Rand index: reference and estimated annotations are sampled uniformly throughout the track, and the induced clusterings are compared by the Rand index [13]; [2]

Normalized conditional entropy: reference and estimated labels are sampled uniformly, and interpreted as samples of random variables Y_R, Y_E, which are computed by estimating the conditional entropy of Y_R given Y_E (*under-segmentation*) and Y_E given Y_R (*over-segmentation*) [9].

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

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 [15]. Melody annotations are represented as time series of fundamental frequency (f_0) values in Hz sampled at discrete points ("frames") in time. An estimated pitch series is evaluated against a reference by computing the following five measures defined in [15], first used in the MIREX 2005 Automatic Melody Extraction evaluation [12]:

Voicing Recall Rate: The proportion of frames labeled as melody frames in the reference that are estimated as melody frames by the algorithm.

Voicing False Alarm Rate: The proportion of frames labeled as non-melody in the reference that are mistakenly estimated as melody frames by the algorithm.

Raw Pitch Accuracy: 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: As raw pitch accuracy, except that both the estimated and reference f_0 sequences are mapped onto a single octave.

Overall Accuracy: The proportion of all frames correctly estimated by the algorithm, where for non-melody frames this means the algorithm labeled them as non-melody, and for melody frames the algorithm both labeled them as melody frames and provided a correct f_0 estimate for the melody (i.e. within half a semitone of the reference).

Prior to computing the measure definitions provided above, both the estimate and reference sequences must be sampled on the same timebase.

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 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 preprocessing 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. We then computed the average relative change across all examples in the obtained dataset for each score.

For the beat detection, chord recognition, 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, 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 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 [14] and compared `mir_eval`'s reported scores to those of MIREX. For pattern discovery, we used the development dataset released by Collins [2] and used the algorithms submitted by Nieto and Farbood [10] for MIREX 2013 to produce estimated patterns. We evaluated the estimated patterns using the MATLAB code released by Collins [2]. The number of algorithms, examples, and total number of scores for all tasks are summarized in Table **??**.

The resulting average relative change for each metric is presented in Table 4. 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

Brian writes this section, which covers the differences in beat.f_measure, boundary.detection, and onset.f_measure

4.2 McKinney's P-score

When computing McKinney's P-score [3], the beat sequences are first converted to impulse trains sampled at a 10 millisecond resolution. Because this sampling involves quantizing the beat times to a 10ms grid, 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 [3]), and as a result, we observe a considerable average relative change for the P-score metric.

4.3 Information Gain

The Information Gain metric [3] involves the computation a histogram of the per-beat errors. The Beat Evaluation Toolbox 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 Other segmentation differences

Brian renames this section and writes it/splits it into different sections.

4.5 Chord stuff

Eric renames this section and writes it/splits it into different sections

4.6 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. 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.

4.7 Weighted Chord Symbol Recall

For reference, we compare our chord evaluation against the results provided from MIREX 2013. Though the **Root** comparison falls well within our accuracy tolerance, the other four comparison rules exhibit non-negligible discrepancies that warrant justification. Based on our current understanding, implementation differences 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

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 Recognition					Melody Extraction				
Root	Maj/min	Maj/min + Inv	7ths	7ths + Inv	Overall	Raw Pitch	Chroma	Voicing R	Voicing FA
0.007%	0.1634%	1.005%	0.483%	0.899%	0.070%	0.087%	0.114%	0.000%	10.095%

"major", the asterisk implies the root is omitted and thus it is unclear whether $D : maj(*1)/\#1 == D : maj1$. 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 4.7. For example, it seems as though $Bb : maj(9)/9$ was *not* excluded from the MIREX evaluation, contradicting the description provided online. This chord alone causes an observable difference between the results of our implementation and previously reported results.

5. DISCUSSION

As seen in the case of evaluating chord estimation systems, it is our hope that an open-source, publicly refined implementation might help resolve independently performed evaluation in the future. Furthermore, the difficulty faced in attempting to match the MIREX chord evaluation precisely clearly demonstrates the need for transparency and community involvement in comparative evaluation.

6. 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] 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.

[3] 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.

[4] 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)*, 2003.

[5] 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.

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

[7] M. Levy. Improving perceptual tempo estimation with crowd-sourced annotations. In Anssi Klapuri and Colby Leider, editors, *Proceedings of the 12th International Society for Music Information Retrieval Conference (ISMIR)*, pages 317–322. University of Miami, 2011.

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

[9] 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.

[10] O. Nieto and M. Farbood. Discovering musical patterns using audio structural segmentation techniques. *7th Music Information Retrieval Evaluation eXchange (MIREX)*, 2011.

[11] 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.

[12] 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 Trans. on Audio, Speech, and Language Processing*, 15(4):1247–1256, 2007.

Chord					Melody				
Root	M/m	M/m + Inv	7ths	7ths + Inv	Overall	Raw Pitch	Raw Chroma	Voicing R	Voicing
0.007%	0.1634%	1.005%	0.483%	0.899%	0.070%	0.087%	0.114%	0.000%	10.09
Segmentation									
NCE-Over	NCE-under	Pairwise F	Pairwise P	Pairwise R	Rand	F@.5	P@.5	R@.5	F@
3.182%	11.082%	0.937%	0.942%	0.785%	0.291%	0.429%	0.088%	1.021%	0.393
Segmentation				Beat					
P@3	R@3	Ref-est dev.	Est-ref dev.	F-Measure	Cemgil	Goto	P-score	CMLc	CML
0.094%	0.954%	0.935%	0.000%	0.703%	0.035%	0.054%	0.877%	0.161%	0.143
Beat			Pattern						
AMLc	AMLt	In. Gain	Pat	Pat	Pat	Pat	Pat	Pat	Pat
0.137%	0.139%	9.174%	???	???	???	???	???	???	???
Pattern									
Pat	Pat	Pat	Pat	Pat	Pat	Pat	Pat	Pat	Pat
???	???	???	???	???	???	???	???	???	???
Pattern				Onset			Source Separation		
Pat	Pat	Pat	Pat	F-measure	Precision	Recall	SDR	SIR	SAR
???	???	???	???	0.165%	0.165%	0.165%	???	???	???

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

[14] J. Salamon and E. Gómez. Melody extraction from polyphonic music: MIREX 2011. *5th Music Information Retrieval Evaluation eXchange (MIREX)*, 2011.

[15] 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.

[16] 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.

[17] 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.

[18] 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.