MIR_EVAL: A TRANPARENT IMPLEMENTATION OF COMMON MIR METRICS

Colin Raffel, Brian McFee, Eric J. Humphrey, Justin Salamon, Oriol Nieto, Dawen Liang, and Daniel P. W. Ellis

Lab ROSA
Laboratory for the Recognition and Organization of Speech and Audio

MARL
NYU Music and Audio Research Laboratory

Abstract

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. We performed a quantitative comparison of `mir_eval` to existing evaluation systems to explore differences in implementation.

Design

`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. 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.

Comparison to Existing Implementations

In order to validate the design choices made in `mir_eval`, we compared the scores it produces to those reported by the evaluation systems used in MIREX. 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. Then, for each reported score, we computed the relative change between the scores as their absolute difference divided by their mean. Finally, we computed the average relative change across all examples in the obtained dataset for each score. The number of algorithms, examples, and total number of scores for all tasks are summarized in the following table.

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

Average Relative Difference of `mir_eval` vs. MIREX

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)

F@3	P@3	R@3	Ref-est dev.	Est-ref dev.
0.393%	0.094%	0.954%	0.935%	0.000%

Onset Detection

F-measure	Precision	Recall
0.165%	0.165%	0.165%

Chord Estimation

Root	Maj/min	Maj/min + Inv	7ths	7ths + Inv
0.007%	0.163%	1.005%	0.483%	0.899%

Melody Extraction

Overall	Raw pitch	Chroma	Voicing R	Voicing FA
0.070%	0.087%	0.114%	0.000%	10.095%

Using `mir_eval` in Python

```python
import mir_eval
# Load in beat annotations
reference_beats = mir_eval.io.load_events('ref_beats.txt')
estimated_beats = mir_eval.io.load_events('est_beats.txt')
# scores will be a dictionary where the key is the metric name
# and the value is the score achieved
scores = mir_eval.beat.evaluate(reference_beats, estimated_beats)
# evaluate() will pass keyword args to the metric functions
scores = mir_eval.beat.evaluate(reference_beats, estimated_beats,
                                f_measure_threshold=0.05,
                                cemgil_sigma=0.02)
# You can also perform pre-processing and compute metrics manually
reference_beats = mir_eval.beat.trim_beats(reference_beats)
estimated_beats = mir_eval.beat.trim_beats(estimated_beats)
f_meas = mir_eval.beat.f_measure(reference_beats, estimated_beats,
                                 f_measure_threshold=0.05)
cemgil = mir_eval.beat.cemgil(reference_beats, estimated_beats,
                              cemgil_sigma=0.02)
```

Using the evaluator scripts

```
> ./beat_eval.py ref_beats.txt est_beats.txt -o scores.json

  ref_beats.txt vs. est_beats.txt
                            F-measure : 0.622
                               Cemgil : 0.362676699474
                 Cemgil Best Metric Level : 0.362676699474
                                 Goto : 0.000
                              P-score : 0.828185328185
    Correct Metric Level Continuous : 0.0328185328185
         Correct Metric Level Total : 0.65444015444
        Any Metric Level Continuous : 0.0328185328185
             Any Metric Level Total : 0.65444015444
                    Information gain : 0.20492902479
  Saving results to:  scores.json

> cat scores.json

  {"F-measure": 0.6216216216216, "Cemgil": 0.36267669947376, ...
```

http://www.github.com/craffel/mir_eval