mir_eval

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

Evaluation Needs

Everyone should use the same code for evaluating their algorithms.
Evaluation Needs

Everyone should use the same code for evaluating their algorithms. In reality, researchers often write their own code. Why?
Evaluation Needs

Everyone should use the same code for evaluating their algorithms. In reality, researchers often write their own code. Why?

- NEMA/MIREX codebase is powerful, but too many moving parts
Evaluation Needs

Everyone should use the same code for evaluating their algorithms. In reality, researchers often write their own code. Why?

- NEMA/MIREX codebase is powerful, but too many moving parts
- Language preferences

Our solution: mir_eval
Evaluation Needs

Everyone should use the same code for evaluating their algorithms. In reality, researchers often write their own code. Why?

- NEMA/MIREX codebase is powerful, but too many moving parts
- Language preferences
- Ease of integration
Evaluation Needs

Everyone should use the same code for evaluating their algorithms. In reality, researchers often write their own code. Why?

- NEMA/MIREX codebase is powerful, but too many moving parts
- Language preferences
- Ease of integration
- Want to understand the metrics
Evaluation Needs

Everyone should use the same code for evaluating their algorithms. In reality, researchers often write their own code. Why?

- NEMA/MIREX codebase is powerful, but too many moving parts
- Language preferences
- Ease of integration
- Want to understand the metrics

Our solution: mir_eval
mir_eval’s Goals

- **Standardized** - mir_eval should implement evaluation metrics as agreed upon by the community, rather than a single researcher.
mir_eval’s Goals

- **Standardized** - mir_eval should implement evaluation metrics as agreed upon by the community, rather than a single researcher.
- **Transparent** - The implementations in mir_eval should make it very clear why the metrics were implemented the way they were. Code should be readable and well-documented.
mir_eval’s Goals

- **Standardized** - mir_eval should implement evaluation metrics as agreed upon by the community, rather than a single researcher.
- **Transparent** - The implementations in mir_eval should make it very clear why the metrics were implemented the way they were. Code should be readable and well-documented.
- **Easy-to-use** - Using mir_eval should be easy whether you’re familiar with Python or not, and should have minimal “start-up cost”.

Compared to NEMA/MIREX:

**Beat Detection**

<table>
<thead>
<tr>
<th></th>
<th>F-measure</th>
<th>Cemgil</th>
<th>Goto</th>
<th>P-score</th>
<th>CMLc</th>
<th>CMLt</th>
<th>AMLc</th>
<th>AMLt</th>
<th>In. Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.703%</td>
<td>0.035%</td>
<td>0.054%</td>
<td>0.877%</td>
<td>0.161%</td>
<td>0.143%</td>
<td>0.137%</td>
<td>0.139%</td>
<td>9.174%</td>
</tr>
</tbody>
</table>

**Structural Segmentation (continued)**

<table>
<thead>
<tr>
<th></th>
<th>F@3</th>
<th>P@3</th>
<th>R@3</th>
<th>Ref-est dev.</th>
<th>Est-ref dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.393%</td>
<td>0.094%</td>
<td>0.954%</td>
<td>0.935%</td>
<td>0.000%</td>
</tr>
</tbody>
</table>

**Onset Detection**

<table>
<thead>
<tr>
<th></th>
<th>F-measure</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.165%</td>
<td>0.165%</td>
<td>0.165%</td>
</tr>
</tbody>
</table>

**Chord Estimation**

<table>
<thead>
<tr>
<th></th>
<th>Root</th>
<th>Maj/min</th>
<th>Maj/min + Inv</th>
<th>7ths</th>
<th>7ths + Inv</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.007%</td>
<td>0.163%</td>
<td>1.005%</td>
<td>0.483%</td>
<td>0.899%</td>
</tr>
</tbody>
</table>

**Melody Extraction**

<table>
<thead>
<tr>
<th></th>
<th>Overall</th>
<th>Raw pitch</th>
<th>Chroma</th>
<th>Voicing R</th>
<th>Voicing FA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.070%</td>
<td>0.087%</td>
<td>0.114%</td>
<td>0.000%</td>
<td>10.095%</td>
</tr>
</tbody>
</table>

Community involvement through issue tracking and pull requests:

- allow unicode filenames in input_output
- Fix boundary detection F-score in extreme cases
- Travis CI + Python3
- replace numpy fft with (faster) scipy.fftpack

http://github.com/craffel/mir_eval
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)
```
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)
```

**Using the evaluator scripts:**

```
> ./beat_eval.py ref_beats.txt est_beats.txt -o scores.json
> cat scores.json
{
"F-measure": 0.6216216216216,
"Cemgil": 0.36267669947376,
"Cemgil Best Metric Level": ...
```
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)
```

Using the evaluator scripts:

```
> ./beat_eval.py ref_beats.txt est_beats.txt -o scores.json
> cat scores.json
{"F-measure": 0.6216216216216216,
 "Cemgil": 0.36267669947376,
 "Cemgil Best Metric Level": ...
```

Using our web API:

http://labrosa.ee.columbia.edu/mir_eval
Where to find us

Code:

http://github.com/craffel/mir_eval

Documentation:

http://craffel.github.io/mir_eval

Web API:

http://labrosa.ee.columbia.edu/mir_eval

Paper: