Python tools for music analysis

Brian McFee

NYU | Center for Data Science

MARL

Why Python?

- Sooner or later, we're all software engineers…
  - … whether we want to be or not
- Python offers a nice balance between power and simplicity
- Active developer community
- Also it's free!

Why not python?

- 20+ years of entrenched legacy code and tools
- Change is hard! Porting code is hard!
  - *Just one more project in matlab...*
- Relatively little support for audio-related tasks
  - *… but I'm trying to change that!*

The (scientific) python ecosystem

(just the highlight reel)

The core: numpy and scipy

- **numpy**
  - numerical array data structures
  - basic mathematical operations

```
>>> import numpy as np
>>> x = np.arange(5)
>>> x
array([0, 1, 2, 3, 4])
>>> x**2
array([0, 1, 4, 9, 16])
```

- **scipy**
  - linear algebra, sparse matrices
  - optimization, DSP, statistics

```
>>> import scipy
>>> scipy.fft(x)
array([ 10.0+0.j   , -2.5+3.441j,
       -2.5+0.812j, -2.5-0.812j,
       -2.5-3.441j])
```

[numpy for matlab users]

Visualization and analysis

- **matplotlib**
  - powerful, but difficult to use properly
- **seaborn**
  - makes matplotlib nice by default
  - adds statistical functionality
- **pandas**
  - time series and statistics (Series)
  - named fields (DataFrames)
  - relational structures (pivots, joins, etc)

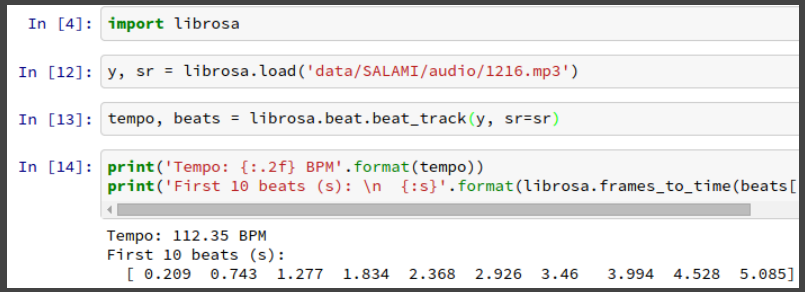
```
>>> import matplotlib.pyplot as plt
>>> fig = plt.figure()
>>> plt.plot(x, x**2)
>>> fig.show()

>>> import seaborn as sns
>>> sns.set()
>>> # [Repeat the plot code above]
```

```
>>> import pandas as pd
>>> data = pd.read_csv('file.csv')
>>> data.describe()
```

IPython and IPython notebook

- Interactive python shell
  - Always use ipython instead of python shell
- IPython Notebook
  - Code *interactively the browser*!
    - images (and plots)
    - video and audio playback
    - widgets and controls
  - Share notebooks on nbviewer
- Jupyter = Notebook - Python



The (musical) python ecosystem

Symbolic data

- **music21**
  - process symbolic score data
  - MusicXML and humdrum
  - extensive documentation

```
>>> from music21 import corpus
>>> sBach = corpus.parse('bach/bwv7.7')
>>> sBach.show()
```

- **pretty_midi**
  - simplified access to midi data
  - easy to read and write
  - documentation / examples

Music information retrieval

- **mir_eval**
  - *How good is my algorithm?*
  - standardized implementation of evaluation metrics
  - stable release, but looking to expand

- **JAMS**
  - JSON storage for annotations
  - collect all annotations for a track in one place
  - standardized schema, validation
  - under (very) active development

librosa: audio processing in python

- Tools to build your own MIR system
  - decode audio
  - STFT, CQT, Mel, chroma, …
  - source decomposition, HPSS
  - onset detection, beat tracking, structural features, …
  - spectrogram visualization

- Provides features for various tasks:
  - structure analysis, similarity, chord and instrument rec, etc

- Some integration with **scikit-learn**

Enough talk, let's code!

Stable version:

```
$ pip install --user librosa
```

Development version (recommended):

https://github.com/bmcfee/librosa/releases

Notebook links

- librosa demo
- Widgets
- Loudness vs IOI

But how do I make it go fast?

- Loops can be slow, just like in matlab

- First step: profile your code!
  - $ python -m **cProfile** -o your_program.profile your_program.py
  - $ **runsnake** your_program.profile

- Solution 1: vectorize your loops!
- Solution 2: **numba** [automagic]
- Solution 3: **cython** [semi-automagic]
- Solution 4: **scipy.weave** [embed C/C++]
- Solution 5: **joblib.Parallel** [use multiple threads]

What are the drawbacks?

- Real-time audio is still a ways out
  - **pyaudio** gives PortAudio bindings
  - latency is still pretty high, but **cython** might help
- Python 2.7 vs 3
  - If you can, just use **python3**
  - **six** can make the transition easier
- Development is rapid, and it can be hard to keep up

Parting words / best practices

- Use **pylint** and **pep8** to avoid simple mistakes
- Use **nose** to handle testing
- Use the **six** module for py2/py3 compatibility
  - even if you only use one version, play nice with others!
- Use **sphinx** and **numpydoc** for documentation
- Use **notebook** for reproducibility, and version control everything!

Thanks!
Questions?

brian.mcfee@nyu.edu
http://bmcfee.github.io/

References / packages

- numpy
- scipy
- matplotlib
- seaborn
- *bokeh*
- *mpld3*
- pandas
- *anaconda*

- ipython

- scikit-learn

- music21
- pretty_midi
- mir_eval
- jams
- librosa
- *pysoundfile*
- *scikits.samplerate*

- cProfile
- runsnake

- numba
- cython
- scipy.weave
- joblib.Parallel

- six
- pylint
- pep8
- nose

- sphinx
- numpydoc