Simple, Open, Music Recommendations with Python

Sam Thursfield

FOSDEM 2023
About me

• Systems software developer @ Codethink
• Musician and music fan
• Former teacher
Playlists (1990s)

❌ Difficult to make
✅ Easy to share

Photo: Alamodestuff, Flickr, CC-BY-NC-ND
Playlists (2000s)

✅ Easy to make
❌ Difficult to share

Photo: Ariszlo, DeviantArt, CC-BY-NC-SA
### Playlists (2010s)

#### Now 2022

<table>
<thead>
<tr>
<th>#</th>
<th>Title</th>
<th>Album</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Running Up That Hill (A Deal A Night)</td>
<td>Hounds Of Love</td>
<td>4:58</td>
</tr>
<tr>
<td>2</td>
<td>Ayee Morshume Be-Reham</td>
<td>Disco Jazz</td>
<td>15:38</td>
</tr>
<tr>
<td>3</td>
<td>I'm Free (Taking Over)</td>
<td>Fog Surrounds Us</td>
<td>3:37</td>
</tr>
</tbody>
</table>

**Recommended**

Based on what's in this playlist

- **Angeleyes**
  - Voulez-Vous

**Other playlists**

- Best of British SKA PUNK
- GalicianTunes Singles 2
- Now 2022
- DIVING IN - FILIP'S RA...
- steezy beats - relaxing...
- Victor Rice Remixes
- Modern Media Mixtape
- VC Music
- Ska Punk Daily Weekly ...
- Music Production
- Bethlehem Casuals /...
Playlists (2010s)

- Easy to make
- Easy to share
- Can generate the playlist for you
Spotify philosophy

• Grow as big as possible ("blitzscaling")
• Pay artists as little as possible
• Optimize for passive listener engagement
• Apply user surveillance and machine-learning to every problem
• All hail the Algorithm
What would the opposite look like?
• Not for profit / DIY
• Encourage building a local music collection
• Link to artist-controlled websites
• Work with open data
Let's get experimenting!
What can we learn from ... Dynamicland?
What can we learn from Git?
Git's core ideas were implemented in a month

1. Well-defined data model: blobs, trees, commits, refs.
2. Multi-call binary: small programs that work together
3. "Porcelain" and "Plumbing" layers

Git's design allows...

- a "polyglot" codebase
- easy extensions
- popular websites built around it
Calliope: the same principle for playlists.

- Data model: everything is a playlist
- Multi-call binary cpe (also has a Python API)
- Build recommendation pipelines as shell pipelines
- Optimized for ease of maintenance over ease of use.

`pip3 install calliope-music`
Core data model

Playlist item

{ "creator": "Artist 1", "title": "Great Song" }

Playlist

{ "creator": "Artist 1", "title": "Great Song" }
{ "creator": "Artist 2", "title": "Banging Tune" }
{ "creator": "Artist 3", "title": "Unpleasant Noise" }

This is JSON Lines data so it can be processed one line at a time.
...based on XPSF

What is XSPF?
- A playlist format like M3U
- XML like RSS
- Pronounced spiff
- MIME type application/xspf+xml

What does XSPF look like?
A very simple document looks like this:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<playlist version="1" xmlns="http://xspf.org/ns/0"/>
<trackList>
  <track><location>file://mp3s/song\_1.mp3</location></track>
  <track><location>file://mp3s/song\_2.mp3</location></track>
  <track><location>file://mp3s/song\_3.mp3</location></track>
</trackList>
</playlist>
```

Calliope's playlist format is documented here.
Demo: playlist manipulation

```
{ "creator": "Artist 1", "title": "Great Song" }
{ "creator": "Artist 2", "title": "Banging Tune" }
{ "creator": "Artist 3", "title": "Unpleasant Noise" }
```

- **Shuffle:** cpe shuffle
- **Export:** cpe export
- Line-based shell processing
- Data-oriented shell processing
What's next?
Content resolution

XSPF is an intermediate format. We expected a new kind of software called a *content resolver* to do the job of converting XSPF to a plain old list of files or URIs.

--- XSPF spec
Demo: content resolution

Three songs:

```json
{"creator": "Kate Bush", "title": "Hounds of Love"}
{"creator": "Madonna", "title": "Holiday"}
{"creator": "Ana Frango Elétrico", "title": "Saudade"}
```

- Resolve locally: `cpe tracker resolve-content`
- Resolve remotely: `cpe spotify resolve-content`
What's next?
Recommendations

big playlist → algorithm → small playlist
Case study: Special Mix

Special Mix generates a 1 hour playlist of discoveries from a specific year.

```
python3 -m calliope_examples.special_mix
```

Ingredients:

1. Listening history: pylistenbrainz
2. Content resolution: beets
3. Track selection: simpleai
1. Listening history

- Use **Listenbrainz** to track music you listen to
- Use **Web Scrobbler** browser extension to submit listens
- Use **pylistenbrainz** and **cpe listenbrainz** to access the data
1. Listening history

cpe listenbrainz listens

```shell
) cpe listenbrainz-history --user samthursfield2 listens \ | from json --objects | first
Updating listens from Listenbrainz server [######################

| listenbrainz.listened_at | 1675368832 |
| listenbrainz.recording_msid | 306525cd-74d3-4acb-b292-8bf300ba6 |
| listenbrainz.artist_msid | 306525cd-74d3-4acb-b292-8bf300ba6 |
| listenbrainz.release_msid | |
| creator | Knobs |
| title | WIW |
| album | Stipple |
| listenbrainz.origin_url | https://knobs.bandcamp.com/album/ |
```
1. Listening history

...choose a year, select by \texttt{first\_listen\_date}: now we have a \texttt{playlist}

\begin{verbatim}
$ cpe listenbrainz-history --no-sync --user samthursfield2 \\ histogram --bucket year | from json | last 5

\begin{tabular}{|c|c|c|}
\hline
# & bucket & count \\
\hline
0 & 2019-01-01 00:00:00 & 6014 \\
1 & 2020-01-01 00:00:00 & 5990 \\
2 & 2021-01-01 00:00:00 & 4239 \\
3 & 2022-01-01 00:00:00 & 6721 \\
4 & 2023-01-01 00:00:00 & 208 \\
\hline
\end{tabular}
\end{verbatim}
2. Content resolution
Beets is the media library management system for obsessive music geeks.

Content resolvers are pluggable and Special Mix can use any...

...now we have a playlist with track URLs and durations.
3. Track selection

The `cpe select` module wraps the Python `simpleai` package.

You define `constraints` for the playlist, then run a `local search` algorithm to try and find a suitable combination of tracks.

No neural network required.
3. Track selection

Music playlist generation by adapted simulated annealing

Steffen Pauws, Wim Verhaegh, Mark Vossen

Philips Research, Prof. Holstlaan 4, 5656 AA Eindhoven, The Netherlands

Abstract

We present the design of an algorithm for use in an interactive music system that automatically generates music playlists that fit the music preferences of a user. To this end, we introduce a formal model, define the problem of automatic playlist gen-
3. Track selection

Each constraint defines a function to score a playlist from 0 to 1.

\texttt{cpe select} searches for the playlist with the highest score given the constraints.

<table>
<thead>
<tr>
<th>description</th>
<th>constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>All different songs</td>
<td>pairs-global(1, n_{\text{max}}, 1, d(v) = {x</td>
</tr>
<tr>
<td>Release in 1980-2001</td>
<td>each-global(1, n_{\text{max}}, 7, [1980, 2001])</td>
</tr>
<tr>
<td>\geq 20% Stevie Wonder</td>
<td>fraction-global(1, n_{\text{max}}, 3, {Stevie Wonder}, .2, 1)</td>
</tr>
<tr>
<td>\geq 10% Seal</td>
<td>fraction-global(1, n_{\text{max}}, 3, {Seal}, .1, 1)</td>
</tr>
<tr>
<td>\geq 10% Peter Gabriel</td>
<td>fraction-global(1, n_{\text{max}}, 3, {Peter Gabriel}, .1, 1)</td>
</tr>
<tr>
<td>\geq 10% Janet Jackson</td>
<td>fraction-global(1, n_{\text{max}}, 3, {Janet Jackson}, .1, 1)</td>
</tr>
<tr>
<td>\geq 10% Mariah Carey</td>
<td>fraction-global(1, n_{\text{max}}, 3, {Mariah Carey}, .1, 1)</td>
</tr>
<tr>
<td>\geq 20% Phil Collins</td>
<td>fraction-global(1, n_{\text{max}}, 3, {Phil Collins}, .2, 1)</td>
</tr>
<tr>
<td>\geq 40% R&amp;B</td>
<td>fraction-global(1, n_{\text{max}}, 5, {R&amp;B}, .4, 1)</td>
</tr>
<tr>
<td>\geq 40% Popular</td>
<td>fraction-global(1, n_{\text{max}}, 5, {Popular}, .4, 1)</td>
</tr>
<tr>
<td>2-3 different genres</td>
<td>cardinality-global(1, n_{\text{max}}, 5, 2, 3)</td>
</tr>
<tr>
<td>Different succ. genres</td>
<td>chain-global(1, n_{\text{max}}, 5, d(v) = {x</td>
</tr>
<tr>
<td>Similar succ. tempi</td>
<td>chain-global(1, n_{\text{max}}, 8, d(v) = {x</td>
</tr>
</tbody>
</table>
Using **local search** to find a solution

Example:

- All songs must be 2 to 4 minutes long.
- The playlist must be 10 minutes long.
from simpleai.search.viewers import ConsoleViewer, WebViewer
from calliope.playlist import Playlist, PlaylistItem
from calliope.select import ItemDurationConstraint, PlaylistDurationConstraint
import calliope.playlist, calliope.select, calliope.shuffle
import sys
MINUTES = 60

constraints = [
    ItemDurationConstraint(vmin=2 * MINUTES, vmax=4 * MINUTES),
    PlaylistDurationConstraint(vmin=10 * MINUTES, vmax=10 * MINUTES),
]

corpus = Playlist([
    PlaylistItem({"calliope.id": "👑", "title": "Amazing Tune", "duration": 2 * MINUTES}),
    PlaylistItem({"calliope.id": "🎸", "title": "Punk Classic", "duration": 1 * MINUTES}),
    PlaylistItem({"calliope.id": "♬", "title": "Lengthy Opus", "duration": 12 * MINUTES}),
    PlaylistItem({"calliope.id": "🌄", "title": "Ambient Noise", "duration": 7 * MINUTES}),
])

viewer = WebViewer()
input_playlist = calliope.shuffle.shuffle(corpus)
output_playlist = calliope.select.select(input_playlist, constraints, viewer=viewer)

calliope.playlist.write(output_playlist, sys.stdout)
sys.stderr.write(f"Total duration: {sum(item['duration'] for item in output_playlist)}\n")
> head 'Special mix 2023-01-20.m3u'

#EXTM3U
#PLAYLIST: Discoveries of 2020

../../Music/Soccer96 - Tactics EP [2020]/01 I Was Gonna Fight Fas
../../Music/Tame Impala - The Slow Rush [2020]/03 Borderline.mp3
../../Music/Vic Ruggiero - On the Ragtime [2009]/09 Don’t Gimme Y
../../Music/Echte Übersee Records_ Finest Latino Ska and Punk Fro
../../Music/KOKOROKO - KOKOROKO [2019]/02 Ti-de.mp3

> head 'Special mix 2023-01-20.m3u.log'
DEBUG:calliope.config:Reading config from /home/pi/.config/calliope/calliope.conf
INFO:root:Using history provider: 'listenbrainz_history'
INFO:root:Using resolver: 'tracker'
DEBUG:root:<class 'calliope_examples.special_mix.special_mix.DiscoveredInTimePeriod'>.setup()
DEBUG:root:Choose one period from: ['2005-01-01 00:00:00', '2006-01-01 00:00:00', '2007-01-01 00:00:00', '2008-01-01 00:00:00', '2009-01-01 00:00:00', '2010-01-01 00:00:00', ...
INFO:root:Query tracks for period 2020-01-01 00:00:00 -> 2021-01-01 00:00:00
DEBUG:calliope.listenbrainz.listens:SQL:
   WITH
     listens_with_track_id AS (...
       SELECT (artist_name || ', ' || track_name) AS track_id, *
Export to music player
Recap: Special Mix

```
python3 -m calliope_examples.special_mix
```

Ingredients:

1. Listening history: pylistenbrainz
2. Local music collection: beets
3. Track selection: simpleai
What's next?
Discussion

Project:

- Code: https://gitlab.com/samthursfield/calliope
- Package: pip install calliope-music
- Documentation: https://calliope-music.readthedocs.io

Forums:

- Beets forum: "Calliope - antisocial music recommendations"
- Metabrainz forum: "Commandline tool for working with Listenbrainz data"

Keep it simple!