Audio content based playlists

You are given a music collection and you need to analyze it to understand what it contains and explore it.

In this task we will create tools for analyzing and previewing music in any given collection. These tools should allow us to:

- Generate automated reports about music collection with some analysis statistics.
- Generate music playlists based on queries by descriptors (tempo, music style, danceability, voice vs instrumental, emotion, etc.)
- Generate music playlists based on queries by track example (tracks similar to a given query track).

We will use the MusAV dataset as a small music audio collection, extract music descriptors using Essentia, generate a report about computed descriptors, and create a simple user interface to generate playlists based on these descriptors.

Music collection

MusAV contains a variety of music, with 2,092 30-second track previews covering 1,404 genres according to metadata used to build the dataset. Download the MusAV dataset from Google Drive. The `audio_chunks` subfolder contains the audio tracks we will use as our music collection.

1. Audio analysis with Essentia

Write a standalone Python script to analyze the entire audio collection and extract the following descriptors for each track:

- **Tempo (BPM)**: you can either use the default settings for the RhythmExtractor2013 signal processing algorithm in Essentia or the TempoCNN ML model.
- **Key**: use KeyExtractor algorithm to compute *key* and *scale*. Because our collection is diverse in genres, use three different profiles for key estimation (`temperley`, `krumhansl`, `edma`) so that we can compare them later. For scale detection, our algorithm is limited to major/minor.
- **Loudness**: use LoudnessEBUR128 to compute *integrated loudness* in LUFS.
- **Embeddings** from Discogs-Effnet (trained on 400 music styles) and MSD-MusicCNN (trained on 50 music tags) models. We will use them for music similarity and they are also necessary for classifiers below.
- **Music styles**: use the Discogs-Effnet model which outputs activation values for 400 music styles. It only works with *discogs-effnet* embeddings.
- **Voice/instrumental** classifier. There are multiple versions of the model (based on different audio embeddings, but they are all pre-trained on the same voice/instrumental

dataset). Use the one based on *discogs-effnet* embeddings because it has highest accuracy.
- **Danceability**: you can either use the signal processing Danceability algorithm or the Danceability classifier model, for which *discogs-effnet* embeddings are expected to produce best results.
- **Arousal and valence** (music emotion): use a model pre-trained on the emoMusic dataset, they appear to have the best performance in our previous evaluations [1]. Regarding the embeddings, *discogs-effnet* performed badly, so we will use *msd-musicnn* instead.

Loading audio:
- Our goal is to avoid loading audio multiple times to optimize computation.
- All our algorithms need mono audio input as a starting point. You can load audio with MonoLoader. However, LoudnessEBUR128 is a special case which needs stereo input. Therefore, if you want to avoid loading audio from a file twice, you can use AudioLoader in combination with MonoMixer and Resample (to 16KHz) for TensorFlow models.

Using ML models:
- The documentation for each Essenia model has code snippets with examples of their use.
- Our goal is to make the computation as fast as possible. Therefore:
  - Instantiate necessary algorithms once, then run computations on each track.
  - Compute required embeddings once, reuse them for many classifier models.
- The classifier models generate predictions on short chunks of audio, resulting in multiple predictions by the same model on different chunks of the same audio track. Therefore, you need to summarize them with averaging (as the simplest strategy). Computation workflow: audio -> embedding frames -> classifier output frames -> Averaging.
- Embeddings for similarity: we will use Discogs-Effnet and MusicCNN embeddings for similarity (see Section 3.2). To compute such similarity, the simplest, lightweight go-to approach is to average the embeddings across time (all chunks) for each track and compute cosine or dot-product similarity between those mean vectors. Therefore, once we obtain all necessary classifier models predictions in the previous steps, we do not need to store the full embedding matrices anymore.

Script requirements:
- Designing your script, keep in mind that you should be able to run this script on any given music collection of any size with any nested folder structure (for example, someone could rerun it on their personal music collection). The script should find and analyze all audio files located inside a given folder. We should be able to re-run your script by easily changing the audio collection path. MonoLoader and AudioLoader support loading any file audio format, but for simplicity let’s limit to searching and loading for MP3 files.
- It is up to you to decide the format and file structure to store the analysis results for the music collection.

- Note that you might encounter analysis errors on some files (unlikely but happens when running analysis on a very large amount of audio tracks) that would end up with an Essentia exception. Your script should be able to skip such errors if they happen and recover analysis without recomputing the tracks that have already been analyzed if it was interrupted.
- It is a nice idea to add a progress bar, for example using tqdm.

Analyze the MusAV audio collection with your script. For reference, it takes 1.25 hours on one of our machines (single thread of Intel(R) Core(TM) i5-4590 CPU @ 3.30GHz).

2. Music collection overview

Once we got all our audio files analyzed, we can now explore our collection.

Write a Python script (or a Python notebook) that generates a statistical report to answer the following questions:

- Which music styles are present in the collection? Plot distribution.
  - As the model predictions are activations, you need to decide what you consider as the final music style prediction for a track. If you want to consider the possibility of multiple styles per track, define some threshold for activation values. Otherwise, if you want to have a single music style per track, use the one with the maximum activation.
  - We have 400 values which may be a challenge to fit in a compact plot. Predicted styles have a parent broad genre category (all style tags have a format \`genre—style\`). Therefore you can instead report distribution for parent broad genres. In any case, also provide full results as a separate TSV file (similar to how we report genre distribution in MTG-Jamendo).
- How fast and danceable is the music? Plot tempo and danceability distribution.
- Which keys/scales are used in the collection? Plot distribution according to the three profiles we used.
- What about loudness? Plot integrated loudness LUFS distribution.
- How is music represented in terms of arousal/valence emotion space? Plot a 2D distribution.
- How much of the music is vocal vs instrumental?

In your written report document, comment on your observations.

- Is this collection diverse in terms of music styles, tempos, tonality, emotion, etc?
- Comment on differences in key and scale estimation using the three profiles (\`temperley\`, \`krumhansl\`, \`edma\`). What is the % of tracks on which all three estimations agree? If we had to select only one profile to present to the users, which one should we use?

- Comment on loudness. Does its distribution make sense? For more about loudness see, for example, https://majormixing.com/how-loud-should-my-master-be/

Find the best way to present results (e.g., select types of plots that efficiently present all information). Some of the Python tools that can be useful to present results:

- https://pandas.pydata.org/
- https://seaborn.pydata.org/
- https://matplotlib.org/

3. Playlist generation

In this part we will create a simple UI for playlist generation based on the results of audio analysis.

We propose that you use Streamlit to create a simple webpage that you can run locally on your computer. Streamlit has different options for input widgets and it will interactively recompute results based on any changes to these inputs.

3.1 Playlists based on descriptor queries

We provide an example of Streamlit code to create playlists based on music style queries: https://drive.google.com/drive/folders/1IxILPD3DbncNBbxpOF589WGMwVXvXf49?usp=share_link

In this demo, we analyzed MusAV with the DiscogsEffnet models. You can extend this code, adding search by the analysis results from the rest of the models, or write your own interface from scratch.

Filtering or ranking by **music style** activations is already implemented in the demo example. Test it with your own analysis results.

For **tempo**, implement search by tempo, where you can specify a tempo range (min to max BPM).

For **voice/instrumental** classifier, implement a binary selector that allows you to show only music with or without voice.

For **danceability**, implement a range selector. In the case of Essentia's signal processing algorithm, its output is within the [0, 3] range. In the case of the classifier, the output value is probability of music being danceable, within [0,1] range.

For **arousal and valence**, the range of values output by the model is within [1, 9]. Implement a range selector. Note: the model is trained on the values within [1, 9] but there might be outliers in predictions.

For **key/scale**, you can have a dropdown menu to select the key and select major vs minor scale. We have estimations done by three different profiles (`temperley`, `krumhansl`, or `edma`). Decide which of them you want to use for the UI.

If you have alternatives ideas for the UI, go ahead and implement them and describe all your decision and implementation steps.

Listen to the playlists

Create various playlists using your UI. There is no easy way to embed an entire playlist on a webpage in Streamlit. In our demo example we embed the top 10 tracks in separate audio players. To listen to the entire playlist, we store it to an M3U8 playlist file. Use an external media player that can reproduce M3U8 files (for example, VLC, Strawberry).

Note that the audio analysis models aren't 100% accurate therefore there will be misclassifications. Discuss some of the issues you observed generating your playlists. Highlight, which analysis models seem to work good for you and which fail.

3.2 Playlists based on track similarity

Apart from the classification tasks, we can also use embeddings we computed before for music similarity. Let's evaluate its quality in a listening test.

Create a separate simple Streamlit demo that allows users to select a *query track* from the collection (for example, by selecting from a list of IDs) and create two lists of 10 most similar tracks, using *effnet-discogs* and *msd-musicnn.* To compute music similarity, use dot product or cosine distance between the embeddings for the query and all other tracks.

Embed music players to show the query and the two lists of most similar tracks, so that you can listen and compare results.

Listen to the playlists

Which embedding produces the best similarity results for you? Why do you think this is the case?

Deliverable

For this task, provide us with

- The code used to generate the features from the audio
- The features that you extracted from the audio
- The code for generating the overview report from the features.
- The code for the user interface (two separate apps, one for queries by descriptors, another for similarity). We should be able to run this on our computer to generate our own playlists.
- A report (~2 pages) describing the decisions you took in all steps, when generating the features, computing statistics overview, building the interface, along with your personal opinion of the quality of the system in terms of its capability to generate playlists. Include your observations on the quality of the extracted features, including examples of good and bad extracted features that you encountered. Report can be larger than 2 pages if you need space to show Figures.

Submission deadline: February 16

If you need help

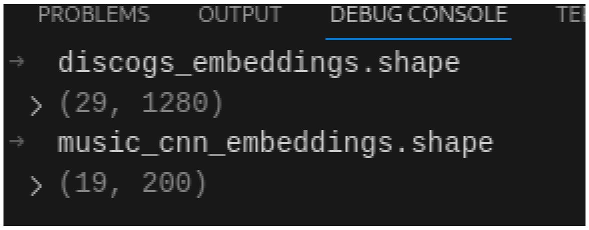
- Comment your question in this Google Doc (so that we can improve the document for everyone).
- Ask me on Slack.

Questions and answers

👉 *Hey, I am getting this warning when running the ML models. I think it's not hindering the process as it seems to be related to the SVM MusicExtractor. Is it safe to ignore it? If so, I was just wondering: how do I suppress it? Cheers*

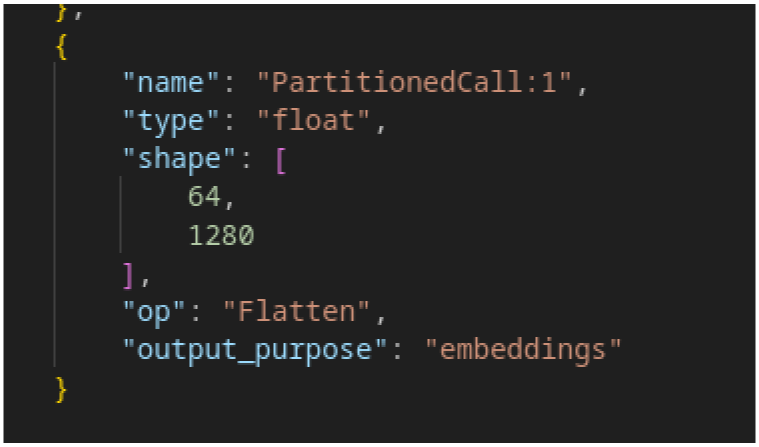
Answer: You can disable warnings:
https://essentia.upf.edu/design_overview.html#logging-in-python

👉 *Why does the output embeddings of discogs-effnet not match with output shape information in the metadata for the models?*



```
→ discogs_embeddings.shape
> (29, 1280)
→ music_cnn_embeddings.shape
> (19, 200)
```

From the metadata json file of Discogs-Effnet:



```
},
{
    "name": "PartitionedCall:1",
    "type": "float",
    "shape": [
        64,
        1280
    ],
    "op": "Flatten",
    "output_purpose": "embeddings"
}
```

Answer:

The resulting shape in your analysis will be the number of frames x dimensionality of embedding (1280 for discogs-effnet, 200 for musicnn). Number of frames depends on the input length. In the documentation we report the batch size x dimensionality embedding. The batch size is the number frames grouped together for parallelization by default: https://essentia.upf.edu/reference/std_TensorflowPredictEffnetDiscogs.html. The model file itself internally works always with 64 frames, but the wrapper algorithm is flexible. TODO Add a note to clarify in https://essentia.upf.edu/models.html#discogs-effnet.

👉 *Could you give a hint about how to compute similarity? The embeddings I computed have the following shape (num_of_frames, num_of_dimensions). Shall I flatten the num_of_frames and compute cosine similarity, or just a dot product?*

Answer: A common approach is to average embeddings across time. See clarification in Section 1 (“Using ML models).