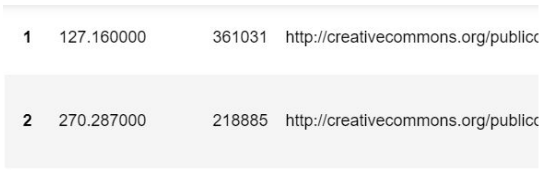
AMPLAB - Freesound Report

Tiange Zhu
Feb.2019

Step1: Create source sound collection

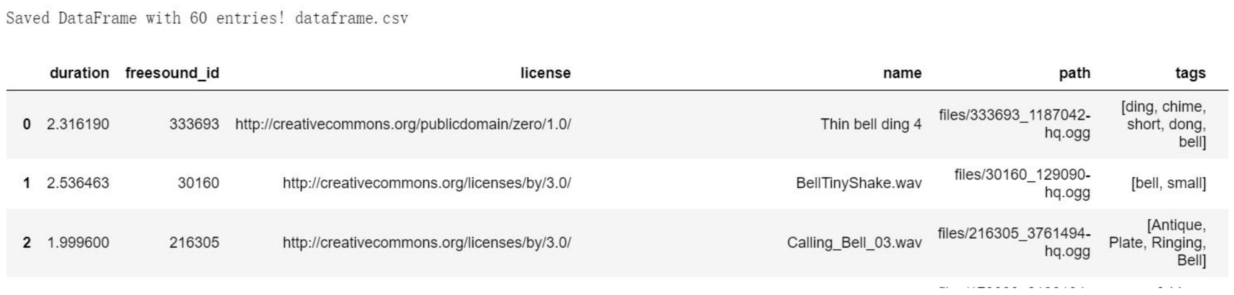
I first replaced API key with mine according to the instruction, then modified the default filter, changed it from [0,30] to [0,20].
I decided to get the sounds of bell, drum and speech as my source samples. I tried to download bell sounds with filter "samplerate: 44100", and "avg_rating": 4", but turns out my query leads to download files way too long, some are even longer than 100s.



1	127.160000	361031	http://creativecommons.org/publicc
2	270.287000	218885	http://creativecommons.org/publicc

Since it's not possible to add multiple filters to one query, I decided to use duration filter for all three sound queries. In the 60 files I downloaded, 20 of them are bell sounds, 20 of them are drum sounds and another 20 of them are speeches. The duration filter is set as "from 1 to 3 seconds". The metadata is stored in ".csv" file, I added 'duration' to FREESOUND_STORE_METADATA_FIELDS, so I could check durations of sounds easily.

Part of the dataframe.csv file is shown below:



```
Saved DataFrame with 60 entries! dataframe.csv
```

	duration	freesound_id	license	name	path	tags
0	2.316190	333693	http://creativecommons.org/publicdomain/zero/1.0/	Thin bell ding 4	files/333693_1187042-hq.ogg	[ding, chime, short, dong, bell]
1	2.536463	30160	http://creativecommons.org/licenses/by/3.0/	BellTinyShake.wav	files/30160_129090-hq.ogg	[bell, small]
2	1.999600	216305	http://creativecommons.org/licenses/by/3.0/	Calling_Bell_03.wav	files/216305_3761494-hq.ogg	[Antique, Plate, Ringing, Bell]

Step2: Analyze source collection and target file

In "anaylze_sound" function, I extracted more features, including using key detection, beat tracking and onset detection algorithms.

Using onset detection for slicing the source files:

In the example, the frame size was fixed, which was 44100Hz / 5 samples = 8820, audio samples were sliced into chunks of 200ms. But I decide to slice the frame size according to their onset positions. I think it would make sliced parts to sound more "complete" individually.
The reference of onset detection is:
https://essentia.upf.edu/documentation/reference/std_OnsetRate.html, there's another algorithm called onset detection, but the input for that one should be spectrogram.

I also noticed that sometimes there's just one onset detected for a source, but there should be at least one start point and another end point for a loop. So for those cases I manually added the end point of audio file to the onset position list. I also ensured it's always an even number to avoid FFT calculation error.

Using beat tracking for slicing the target file:

I saw some empty values in the "beat position" column of df_source when I used beat tracking algorithm for analyzing source files. Then algorithm is from: https://essentia.upf.edu/documentation/reference/std_BeatTrackerDegara.html.

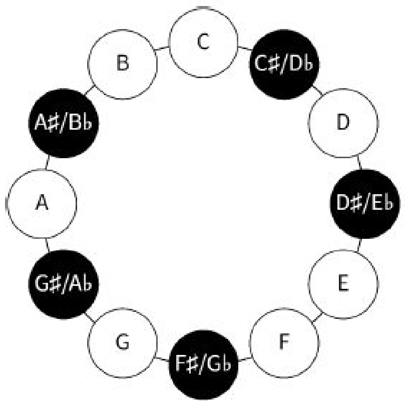
By the definition on Essentia website, the fastest tempo to detect is [60, 250] bpm, and the slowest tempo to detect is [40, 180] bpm. Thus, it's not surprising: some of my source samples are speech, which do not include beat, some are percussion sounds, just containing one hit of drum or bell, which clearly does not have a tempo. Some are with few kicks, while the duration of audio samples are also limited to 0~3s, which makes them too short to have a set of detected beat positions.

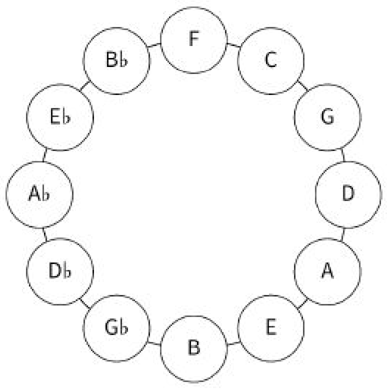
Although beat tracking is not a good choice for slicing source files, it can be used for chunking the target file. The target file is way longer than source files, which makes it more possibly to have a bpm within the preferred tempo range. Chunking by beat positions keeps the rhythmic information of the target file, I suppose it would work well with the source files sliced according to onset positions.

Using key detection for filtering source samples:

As for keys, they are related to pitch. I translated key to numbers, found the key number of target file, then filter the source files by only keeping the source samples that have same key or have the two most similar keys. For example, if the key of target file is E, then I keep the source files that have a key of E, B, and A.

I used music theory taught by Rafael to accomplish this part. As pictures shown below, the left one shows the relationship between flat and sharp, the right one shows the similarity relationship between keys.





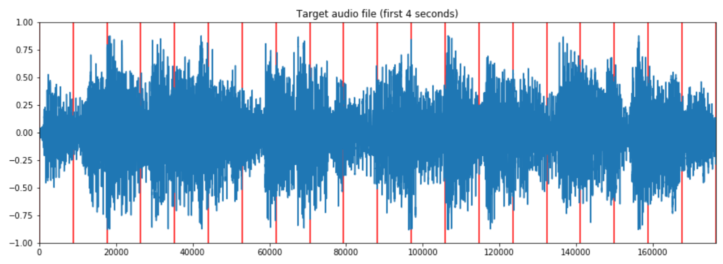
I also tried to calculate danceability, bpm and other things, but in the end I can't think of a good way of using them for audio mosaicing.

Step 3: Reconstruct target file with source collection frames

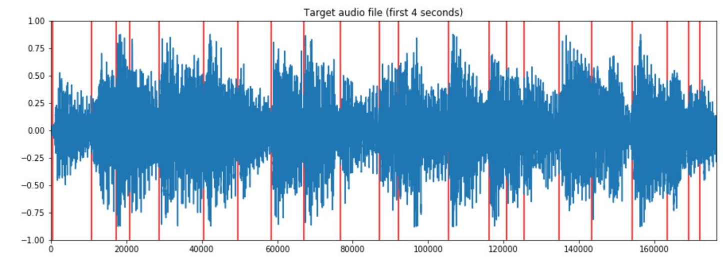
In this step, the program used k-nearest neighbour algorithm to find the most similar frame according to the mfcc feature. The "similar" here is defined as timbre similarity. I applied randomization before the function return the most similar frames, I only kept 5 out of 10 most similar frames, thus there's a big chance that the "most similar frame" returned by "chose_frame_from_source_collection" function is not the most similar one.

Step 4: Analyzation for the target sound and its reconstruction

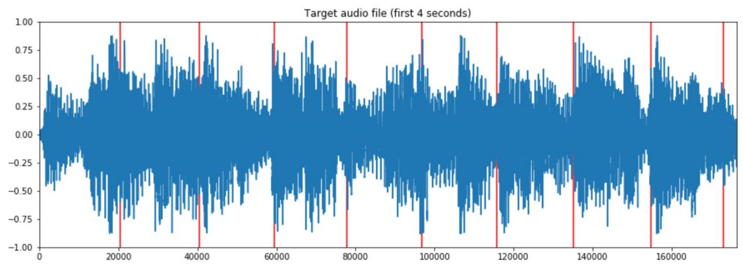
I first chose https://freesound.org/people/NoiseCollector/sounds/82823/ as my target file. It's a flamenco style guitar solo.
By default, the target sample was chunked by frame size = 8820, which looked like below.



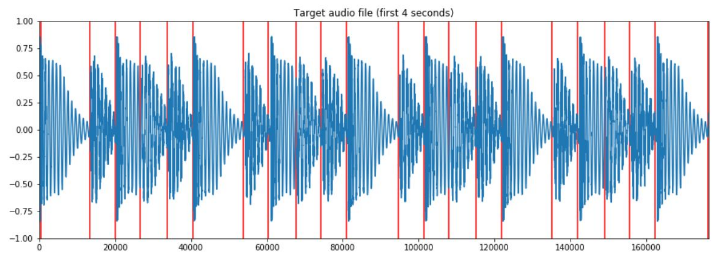
When I use onset detection on target sample, it works, the slices looked like below:



But when I use beat tracking algorithm on the target sample, it's not working well:

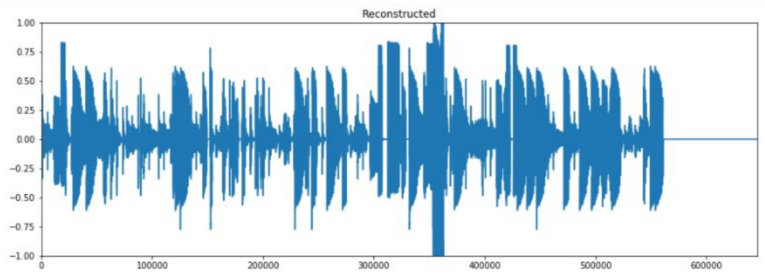


I tried to use beat tracking for slicing the example file ("262350__stereo-surgeon__grinder-drum-loop.wav"), to make sure that my beat tracking algorithm is actually working:



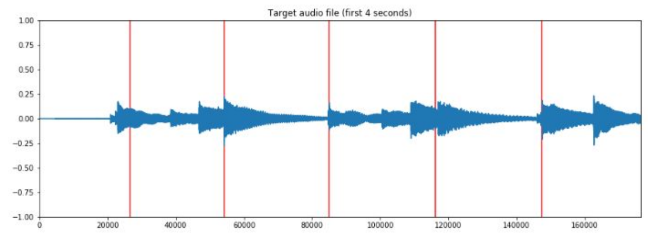
Then the problem should have something to do with the character of this chosen sample, since it has quick attacks, with no percussion sounds in the background. In this specific case, beat tracking algorithm failed to detect the tempo and the beats. However, onset detection still works well with the sample, since it's less "musical" compared to the beat tracking algorithm, as it's more related to the detection of energy change.

Finally, I decided to apply onset detection on both target and source files for slicing, then the final output sounds pretty good. The plot of reconstructed audio sample is shown below:

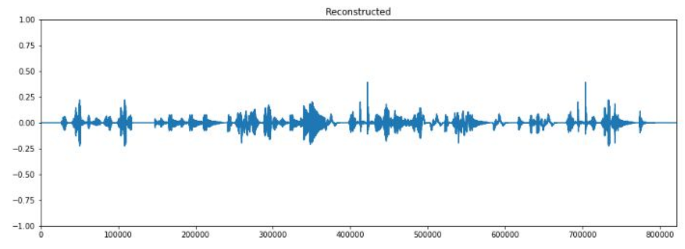


Then I chose another target file, https://freesound.org/people/KAMARIN/sounds/417934/, which is a piece of lullaby from music box, it has a clear rhythm thus the beat tracking algorithm works on this one. This time I applied onset detection for slicing source files, beat detection for slicing the target file, then filtered source files with their key.

The target file slicing process is shown below:



The plot of reconstructed sample is shown below:



It met my expectation. I attached the generated audio file in the zip file.