Automatic Track Mashups/Transitions for Playlists

Ben Harrington
benmharrington@gmail.com

ABSTRACT

Using key detection and tempo matching, as well as splitting tracks into their individual stems, this program takes a playlist of songs as input, and returns an audio file containing each track as well as transitions between each track, much like one would hear a DJ play at a club or a bar. Each track is tempo matched and the transition will consist of a mashup between sequential tracks in the playlist. The program determines an average tempo for all tracks within the playlist, then tempo-matches each track to this average. Due to this feature, playlists with similar tempos or even genres will be simpler for the program to process cleanly and give a more cohesive resulting file. In addition to tempo matching, pitch and key detection is used to determine how best to use transitions between tracks, as well as using stems to ensure that the mashup transitions between tracks sound smooth and cohesive.

1. PROCESS

When creating a playlist for an event or a party, it would be useful to have an automated feature to be able to ensure that transitions between tracks are continuous and flow smoothly to the next song so the music in the playlist never truly “ends”. Ideally, the program would create a transition from a track, to a mashup, to the next track as seamlessly as possible without causing the music to end until the playlist had played all the way through.

1.1 Selecting data sets

The first task for this project was selecting data sets. These data sets were used for testing purposes throughout the development of the project and generally represent music that this program would be used for. As such, some entire genres, such as jazz or classical music, were excluded for testing purposes despite their potential to run smoothly depending on the artist.

1.2 Song analysis

After selecting data sets, each song within the given set was analyzed by the software. The first and foremost target of analysis was the tempo of each song. Using downbeat tracking [1-3] as well as using inspiration from neural networks [4], the tempo was the first step in extracting data from each song. After this, the software determined the beats & downbeats of each track to ensure beat matching between segments.

1.3 Breakdown of each song into stems

Using spleeter [9] and its features, the program split each song into its respective stems in order to better create mashups with upcoming tracks. If the key was properly detected, the instrumentals and vocals could be blended together, and if key detection has failed or was too dissimilar to the next track, then one track’s drums could be blended into the stems on the following one.

1.4 Creating the mashups

The penultimate step of the process involved creating mashups from the end of the current track into the beginning of the next. If the key was properly detected, the instrumentals and vocals could be blended together, and if key detection has failed or was too dissimilar to the next track, then one track’s drums could be blended into the stems on the following one. [10, 11].

1.5 Concatenating songs & mashups

Once the mashups were created, the simple matter of concatenating one song into a mashup into the following song was relatively straightforward. Crossfading between the original tracks and the mashups could be a useful tool in ensuring that the transition between the three states is blended well and not overly jarring.

2. RESOURCES

To complete these steps, I used Python, as a number of Python libraries exist that are helpful for analyzing and manipulating audio. Librosa was used for beat tracking, tempo analysis, and timestretching. Spleeter and its internal audio loader were used for loading a track in stereo, and using library functions to split each track into multiple stems of itself. Madmom [16] was used for downbeat tracking to ensure better blending of mashups.

3. CURRENT METHOD

As a number of tracks N are initially loaded into the program, they are stored both as stereo tracks and as mono tracks. This is to ensure that both spleeter and librosa are able to function as spleeter uses stereo tracks only, and librosa can accurately use the beat tracking and tempo matching functions, which uses mono data rather than stereo. Each track is run through the librosa beat_track function, which returns a list of timestamps (where beats occur) as well as the estimated tempo. In addition, the tracks are run through the madmom downbeat feature detection, which tracks beats as well as downbeats of a bar. After doing a number of auditory tests, librosa generally seemed to be more accurate when it came to accurately tracking beats, but didn't contain a function that tracked downbeats. In order to get around this, when a downbeat is selected it runs through the beats tracked by librosa and selects the timestamp closest to the chosen madmom downbeat. This ensures that the beats are reasonably accurate while still determining whether a beat is a downbeat or not.

After calculating the mean tempo, each track is given a stretch value sv, where $sv > 1$ represents an increase in tempo to match the mean tempo and $sv < 1$ represents a decrease in tempo to match the mean. All of this information, along with track length and tempo, is stored in a data structure *song_data*.

From each track, a segment is taken from both the beginning and the end of the stereo tracks. The length L of this segment is determined by using the stretch value where

$$L = \text{m\_length} * sv,$$

to ensure that after the tracks and segments are timeshifted the mashups between them are all of similar length, where m_length corresponds to the chosen segment time. Segments from the beginning of the tracks are taken from the first recognized downbeat to the sample L seconds after this downbeat. Segments from the end are taken from the the closest downbeat of length L from the final recognized beat of the track to the sample L seconds after this downbeat. This ensures that the segments are not only time stretched to the same tempo, but both begin on a downbeat so mashups are not noticeably offbeat and are also matched at the same point in the measure.

At this point these segments are run through the spleeter library, using the spleeter:4stems model. Each of these stems is converted to mono and time stretched according to their respective sv coefficients. The program creates N-1 mashups by summing together the end segment of a track's drum stem with the beginning segment of the following track's vocal stem, bass stem, and 'other' stem. These mashups are stored in a separate list.

The final concatenation writes all the separate components into a numpy array, and uses the scipy [17] wav.write function to convert it to a readable audio file. The sections contain short crossfades which are currently set to the length of 1 beat of the mean tempo, but this parameter can be adjusted to taste. These crossfades are in place in order to ensure that the transitions are not too harsh.

4. DIFFICULTIES/POSSIBLE FUTURE WORK

Some general assumptions have been made when writing this program. The first assumption relies on all tracks being in 4/4 time. This assumption helps with the speed of the donwbeat tracking function and ensures that each downbeat is tracked with more accuracy. Additionally, creating mashups between tracks with various time signatures presents a number of problems that are not currently addressed in the code. Another assumption made is that each track passed into the playlist will be successfully analyzed by one of the beat tracking algorithms. Not only would a track that did not pass this criteria not have noticeable beats to blend in with other tracks, its tempo would not be detected and it could not be time stretched to match the mean tempo of the remainder of the tracks.

While mashups are currently beat-matched, it is difficult to anticipate which beat of a measure the beat times in each track belong to. This leads to mashups where mashups are generally beat matched, but they do not both start at the first measure of a bar, and are offset by 1-3 beats. The current method uses two separate beat tracking algorithms, one of which (madmom) denotes which of the beats it estimates is a downbeat. Using this method the program is able to more accurately match up songs and create more feasible matchups, but this additional step creates a significant toll on the runtime of the overall program. Generally, this downbeat tracker adds about 1 minute of runtime per track. Evidently when scaling up to a large amount of tracks run through the program, this significant leap in runtime should be taken into account when deciding if the improved accuracy of downbeat tracking is worth the extra time.

It should be noted that in its current form, there is no functional key tracking or pitch shifting being done to any of the tracks. A significant amount of testing was put into pitch shifting tracks in order for mashups to sound more interesting, but the results were mediocre at best. The main complications of key tracking and pitch shifting stemmed from the accuracy of the key detection algorithm, as well as the jarring nature of more aggressive pitch shifts. This likely stems from the fact that many of the songs being tested were of the electronic and hip hop genres, leading to confusion from the algorithm as songs from these genres don't always follow general chord progressions from western music. The other main issue was that once songs are pitch shifted more than a couple semitones, combined with the fact that they had been timestretched as well, their audio quality is significantly compromised.

If future work is done on this project, the first goal would be to find a more accurate method of key detection for each track, as well as better methods for pitch shifting each track to match a global key. Another possible avenue to explore would be to rearrange tracks so that one track's key would be the dominant key of the following track, and so on and so forth.

In addition to the improved key detection and pitch shifting, more robust beat tracking methods would be helpful in ensuring smooth mashups as transitions between tracks. In its current state, if the beat tracking algorithms find an incorrect beat in either of the tracks to be mashed, the entire mashup between the two is completely offbeat and the result is grating.

5. CONCLUSION

Overall, the project accomplished a rudimentary form of what it set out to do, however, the end result gives a glimpse into what could be done with more powerful beat tracking and key detection algorithms. For the most part, spleeter does a reasonable job of separating tracks into their respective stems so when certain stems are added to the drums of another track on beat, the mashup is a reasonable representation of the two songs remixed together. When spleeter is unable to provide accurate stems, the audio quality of the mashups drops significantly and the transitions from mashups to tracks is not as smooth as possible.

In general, if tracks are inputted that have similar tempos and easily trackable beats, the mashups are passable and provide smooth transitions between tracks. When tracks have long intros or outros with undetectable beats, or vary wildly in tempo, the project has more difficulty providing acceptable mashups.

6. REFERENCES

[1] M. Davies, M. Plumbley, and D. Eck. "Towards a musical beat emphasis function." In 2009 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics, pp. 61-64. IEEE, 2009. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/5346462. [Accessed: 14-Oct-2019]

[2] M. Davies, and M. Plumbley. "A spectral difference approach to downbeat extraction in musical audio." In 2006 14th European Signal Processing Conference, pp. 1-4. IEEE, 2006. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7071189. [Accessed: 14-Oct-2019]

[3] B. McFee, H. C. Crayencour, J. Pablo Bello, M. Fuentes, S. Essid,. "Analysis of Common Design Choices in Deep Learning Systems for Downbeat Tracking." In ISMIR, pp. 106-112. 2018.

[4] H. Schreiber and M. Müller. "A Single-Step Approach to Musical Tempo Estimation Using a Convolutional Neural Network." ISMIR. 2018.

[5] A. Lykartsis, M. Stein and R. B. Gebhardt. "A Confidence Measure For Key Labelling." In ISMIR, pp. 3-9. 2018.

[6] R. Mahieu. "Detecting Musical Key with Supervised Learning." (2017).

[7] A. Faraldo, F. Hörschläger, M. Le Goff, P. Herrera, P. Knees, R. Vogl and S. Böck. "Two Data Sets for Tempo Estimation and Key Detection in Electronic Dance Music Annotated from User Corrections." In ISMIR, pp. 364-370. 2015.

[8] F. Korzeniowski, G. Widmer. "Improved chord recognition by combining duration and harmonic language models." 2018.

[9] https://github.com/deezer/spleeter

[10] M. Davies, P. Hamel, K. Yoshii, and M. Goto. "AutoMashUpper: Automatic creation of multi-song music mashups." IEEE/ACM Transactions on Audio, Speech, and Language Processing, vol. 22, no. 12, pp. 1726-1737, 2014. [Online]. Available: https://ieeexplore.ieee.org/document/6876193. [Accessed: 14-Oct-2019]

[11] H. Ishizaki, K. Hoashi, and Y. Takishima. "Full-Automatic DJ Mixing System with Optimal Tempo Adjustment based on Measurement Function of User Discomfort." In ISMIR, pp. 135-140, 2009. [Online]. Available: https://ismir2009.ismir.net/proceedings/PS1-14.pdf. [Accessed: 14-Oct-2019]

[12] J. Robert, "jiaaro/pydub", GitHub, 2019. [Online]. Available: https://github.com/jiaaro/pydub. [Accessed: 16- Oct- 2019]

[13] "Mixing two audio files together with python", Stack Overflow, 2010. [Online]. Available: https://stackoverflow.com/questions/4039158/mixing-two-audio-files-together-with-python?noredirect=1&lq=1. [Accessed: 13- Oct- 2019]

[14] T. Giannakopoulos, "pyAudioAnalysis: An Open-Source Python Library for Audio Signal

Analysis", PLoS ONE, vol. 10, no. 12, 2015 [Online]. Available: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0144610. [Accessed: 14- Oct- 2019]

[15] R. Sagar, "Beginner’s Guide To Building A Song Recommender In Python", *Analytics India Magazine*, 2019. [Online]. Available: https://analyticsindiamag.com/beginners-guide-to-building-a-song-recommender-in-python/. [Accessed: 16- Oct- 2019]

[16] madmom: a new Python Audio and Music Signal Processing Library. Author = Bock, Sebastian and Korzeniowski, Filip and Schluter, Jan and Krebs, Florian and Widmer, Gerhard. https://github.com/CPJKU/madmom

[17] https://github.com/scipy/scipy