Automatic Hip-hop dance playlist ordering

Nathan Monhart

Background

I am a first-year master's student in robotics (4th grade) with a bachelor's in engineering (with a computer science orientation), so I did not know anything about music processing before this project. I did this project in my free time, with no supervisor or reference person so please forgive my imprecisions if there are any. Diving into a completely new field without guidelines is very time-consuming so I didn't have the time to explore all that I first wanted.

I think that this project presents interesting approaches that could benefit music processing and hip-hop dance communities because hip-hop is unfortunately almost not studied in the *MIR* field.

I. Introduction

Automatic playlist generation is a very important part of streaming services and relies on very performant recommendation algorithms. However, the order of the songs in this kind of playlist is sometimes far from optimal. Two consecutive songs can belong to quite different and not very compatible music genres, have contradictory lyrics, and have very different energies (we can pass from a calm to an aggressive song).

The other motivation for this project is that hip-hop dancers often train on the same mixtapes. A mixtape is a continuous mix of several songs with smooth transitions between them. Training with mixtapes rather than automatic playlists helps the dancer to stay in the same energy and allows him to not stop dancing at each change of song. But training on the same mixtapes results in lassitude and lack of motivation during training. Automatic mixtape generation could solve this problem. By the way, automatic mixtape generation could also be used for parties where no DJ is available (to avoid interruptions at each song change and to have a coherent order in the songs).

The goal of this project is to explore different tools that could be used to order song playlists and to provide a method to order the songs of a given playlist using these tools (but not to provide a fully operational algorithm). A special focus will be put on hip-hop dance playlists because they have some particularities that do not apply to other music genres (e.g., they principally rely on drums). However, the core principles and some tools that will be developed can be applied to other music genres.

The method used in this project consists in extracting some relevant information about the songs and ordering them using this information. The extracted information will be the key, the mode, the tempo, and the kick and snare events of the songs. We assume that a recommendation algorithm gives us n songs and the goal of the method is to find the best combination of $N \leq n$ songs. This "optimal" song combination can be used to automatically generate a mixtape, but this will not be covered in this project.

I.A Tools

Only Machine Learning models were used for prediction in this project. The reason is that I am not familiar enough with Deep Learning to use it properly. The model's hyperparameters were found with a grid search that works with cross-validation to find the hyperparameter combination that maximizes a score associated with each model.

The code is written in Python using *Scikit-Learn* for Machine Learning and *Librosa* for specific Music Processing tools [1]. The Notebook associated with this project can be found on GitHub: https://github.com/NathanMonhart/Automatic-Hip-hop-dance-playlist-ordering.git

II. Tempo Extraction

Tempo plays a significant role in hip-hop dance music; the dancer must adapt his dance style to it. A fast tempo results in a high-energy dance style with impressive moves and a slow tempo in a more subtle and smooth

one. For this reason, it is important for two consecutive songs to have a similar tempo.

During the whole project, tempo and beat events were extracted using a *librosa* feature that uses Dynamic Programming for beat tracking. More details about this method can be found in the work of Daniel P.W. Ellis [2].

III. Key and Mode Detection

Key is important in the song order in a mixtape because the transition between two consecutive songs that differ from more than one semitone often sounds dissonant. The mode is also important because it is linked to the mood of a song (major songs are generally associated with happiness-like emotions while minor songs tend to be associated with sadness).

Key and mode detection was achieved using chromagrams. A chromagram is a specific feature representation of an audio signal that represents the pitch distribution across time. It is computed using pitch-based logarithmic spectrograms $C(n,p)$. The following equations are taken from the *Fundamentals of Music Processing* textbook by Meinard Müller [3].

$$C(n,p) := \sum_{\{p\in[0:127]:p \bmod 12=c\}} Y_{LF}(n,p)$$

where $Y_{LF}(n,p)$ is given by

$$Y_{LF}(n,p) := \sum_{k\in P(p)} |X(n,k)|^2$$

with $X(n,k)$ the spectrogram of the audio signal

III.A Dataset

The *Million Song Dataset* was used for training [4]. It is a dataset containing one million popular annotated songs. Since the whole dataset has a size of 280GB, only a subset of 10 000 songs was used here (provided by the authors of the dataset). The features of interest were the chroma features, the key, and the mode of the songs. The data was already normalized.

One problem with the dataset is that the key and the mode are given with some uncertainty (a score between 0 and 1). Data with a confidence score below 0.75 were eliminated from the dataset to ensure better reliability. It however decreases the number of samples from 10 000 to ≈1500.

Two approaches were used to predict the mode and the key of the songs with different kinds of input data. These 2 approaches were using SVM and Random Forest for prediction (with grid search for hyperparameters).

III.B Per-beat Chromagram and PCA

The first method uses chroma features averaged at each beat. They are not given in this format in the dataset, so I had to combine the given chroma feature's start time with the start time of each beat to average chroma features on each beat. To ensure that all the samples have the same size, only the features for the last 100 beats are selected, which gives us $[100x12]$ dimensional data for each of the ≈1500 samples. PCA is then applied to the data to reduce its dimensionality. However, after a 90% variance retaining PCA, the data has still ≈500 features for each sample. SVM and Random Forest are then used for the prediction part.

III.C Chroma Features averaging

The second method was proposed by Robert Mahieu [5]. It averages the chroma features over the song to predict their key and their mode resulting in $[1x12]$ dimensional samples. SVM and Random Forest are then used for the prediction part.

III.D Results

We observe that averaged chromagrams perform significantly better than per-beat chromagrams. This result is even more pronounced with Random Forest, probably because per-beat chromagrams have a very high dimensionality (SVM tends to perform well with high-dimensionality data). This is good news since per-beat chromagrams are computationally heavier than averaged chromagrams and less reliable because they rely on beat detection.

The goal here was to compare these 2 approaches, however, more details on the averaged chromagrams method can be found in the work of Mahieu Robert [5]

Classifier	Model	Input	Test accuracy (%)	Test Recall (%)	Test Precision (%)
Mode	C=1 gamma=0.001 kernel = rbf	Per-beat chromagram	81.5	50	40.8
Key	C = 10 Gamma = 0.001 Kernel = rbf	Per-beat chromagram	64.5	60.3	63.3
Mode	C = 1 Gamma = 0.001 kernel = rbf	Averaged chromagram	74.7	50	37.4
Key	C = 100 Gamma = 0.001 kernel = linear	Averaged chromagram	76.8	74.2	75.6

Table I: Key and mode detection, SVM methods comparison

Classifier	Model	Input	Test accuracy (%)	Test Recall (%)	Test Precision (%)
Mode	Max depth = 1	Per-beat chromagram	81.5	50	40.8
Key	Max depth = 20	Per-beat chromagram	51.5	40.0	49.1
Mode	Max depth = 75	Averaged chromagram	75.3	51.2	87.6
Key	Max depth = 20	Averaged chromagram	71.4	66.6	73.2

Table II: Key and mode detection, Random Forest methods comparison

IV. Drum Detection

Unlike most other music genres, one of the particularities of hip hop is that its subgenres differ by their drums and not by their instrumental part. For instance, *boom bap* is defined by a specific kick/snare pattern over which any type of instrumental can be added (soul, jazz, classical music…). It is the same thing for most other hip-hop subgenres, including hip-hop dance music.

The idea behind including drums detection in the playlist ordering algorithm is that it is not appropriate to randomly switch from one hip-hop dance music subgenre to another. Each subgenre has its own dynamic and dancers have to adapt their style to each of them. For this reason, it is better to have smooth transitions between these subgenres.

For the purpose of this project, drum detection has been simplified under the assumption that the drum content of a hip-hop music subgenre is defined by its kick/snare pattern. It is a reasonable assumption because hip-hop dancers are just using these drums to set the rhythm of their dance (other drums like hi-hats or cymbals are in general too weak or unpredictable to have an impact on the dancer). The following method will analyze snare and kick events at each beat fraction using Machine Learning in order to compute a drum similarity score between 2 songs. This score will be taken into account in the playlist ordering algorithm.

Using beat as a time scale instead of time or sample stamps is a way of getting rid of tempo differences between 2 songs. In this way, 2 songs that have the exact same drum content, but a different tempo will have the same temporal representation.

IV.A Data

The data used by the Machine Learning algorithm to detect kick and snare events at some beat locations will be the normalized FFT of some drum's audio signal. This is because the spectral content of a kick can be easily differentiated from the spectral content of a snare. Different steps are required to obtain this data.

Drums audio signals (kicks, snares, hi-hats, cymbals, claps…) have been obtained using some hip-hop *FL Studio* drum packs. Once they were extracted, target drum signals were randomly combined with some noise for data augmentation (percussive noise; cymbals, hi-hats, claps,… and ambiance noise; white noise, vinyl crackling sounds,…) making sure that no more than one target drum type was present in the signal at a time. Then, the FFT of the resulting audio signal was computed. This process ended up with 2 distinct datasets. One containing the FFT of each sample with a Boolean telling if a kick is present in each sample. The other with the FFT of each sample and a Boolean telling if a snare is present in each of them. This process is summarized in fig 1. Once the kick and snare datasets have been generated, PCA can be applied for dimensionality reduction (data is already normalized). We ensure that PCA retains around 95% of the variance, reducing the dataset to ≈20 features.

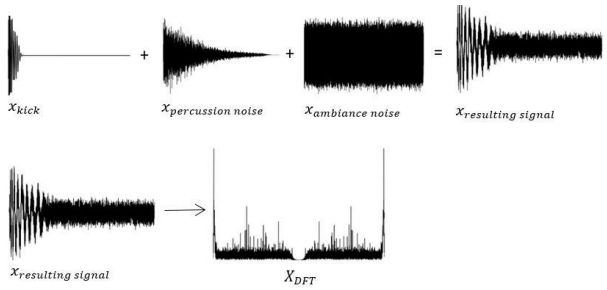

Fig 1: example of a sample generation for the kick dataset

The main advantage of this method is that we have control over the dataset. We can choose specific drum and noise signals and combine them in different ways to produce the samples.

IV.B Detection

The prediction was done using SVM and Random Forest with a cross-validation grid search to find hyperparameters.

Some pre-processing is necessary in order to perform prediction on actual songs. The first step is to separate the drums from the vocals and the instrumental part. This step is done using *Spleeter* [6] - a *Tensorflow*-based python library provided by *Deezer* - to separate audio

files into different components (drums, vocals, instruments,…). The second step consists in dividing the audio signal according to beat events. To do so, we can use beat detection to divide the audio signal into samples at every beat event and then divide these samples into smaller fractions. Kick and snare detection are then applied on the DFT (reduced with PCA) of these samples to have kick and snare events at each beat fraction. By this way, the drum content of a song is independent of its tempo. The result of his procedure is illustrated in fig 2.

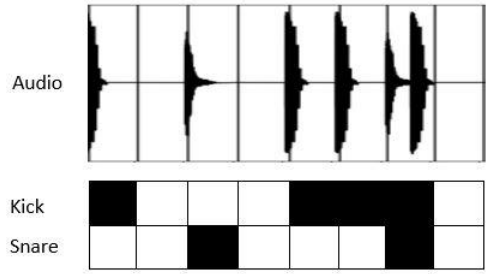

Fig 2: ideal representation of kick and snare event detection on a beat-based time grid

However, the main problem with this technique is that it is difficult to achieve a good beat resolution. Indeed, the method analyzes the spectral content of the signal at each beat fraction. But some drums (especially with reverb) can last a whole beat, so their spectral content will be present at each beat fraction. Since we are only interested in the drum's onsets, this can be a problem.

One way to address a part of this problem is to combine drum prediction with onset detection. If no onset is detected at a certain beat fraction, we assume that no kick and no snare are present in this beat fraction. Onset detection is performed using *librosa*, more details on onset detection can be found in the book of Meinard Müller [3]. This procedure works obviously better when there are not a lot of drums other than kick and snares present in the audio signal. If it is not the case, onset detection will detect all these other drums and so onset correction will not be very efficient.

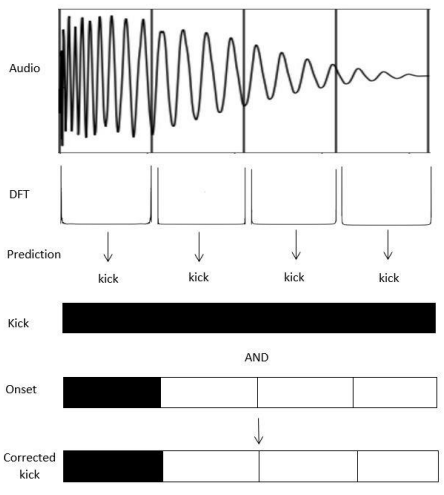

Fig 3: kick and snare event detection with onset correction

IV.C Results

Drum detection works extremely well on the test set. This is probably due to the fact that we can generate the dataset automatically, so we can ensure that we have enough data for each drum that we want to detect.

The performances on the kicks are higher than the performances on the snares probably because they are easier to detect. Indeed, they have unique low-frequency components that are not shared with other drum types or common sources of noise.

Classifier	Model	Test accuracy (%)	Test recall (%)	Test precision (%)
kick	SVM (C=1000, gamma=0.001, kernel = rbf)	98.6	98.5	98.6
kick	Random Forest (max depth=20)	98.6	98.6	98.6
snare	SVM (C=1000, gamma=0.001, kernel = linear)	89.4	89.5	89.5
snare	Random Forest (max depth=30)	95.0	94.9	95.1

Table III: Kick and snare detection methods comparison

However, snare detection doesn't always work well outside of the test set, for actual songs (fortunately, it is not the case for kick detection). In particular when a lot of other drum types are present in the audio signal. In this case, the model often predicts that no snare is present in the audio signal at all. The problem is probably due to the quality of the dataset because the prediction works well for simple audio signals (like the ones present in the dataset) but not for more complex ones. But it is difficult to have a dataset that is representative of real situations. In addition, the drum's separation from *spleeter* often adds a characteristic noise that is not taken into account in the dataset.

Onset correction brings significant improvements to drum detection. Onset detection is very reliable, but a bit of calibration is necessary in order for it to work well. Indeed, onsets are often detected with a certain offset. A 15% shift of the detected onsets is therefore realized to correct that. As said before, onset correction is particularly useful for simple audio signals but shows its limitations with more complex audio signals.

IV.D Similarity computation

Once the drum content at each beat-fraction frame can be obtained, it is possible to compute a drum similarity measure between 2 songs. The method proposed in this project consists of several steps. The first step is to compare separately the kick and the snare content of the 2 songs with the XNOR operator. The XNOR operator is applied between the 2 kick and snare sequences using

cyclic permutations. By using cyclic permutations, the score is independent of time shifting.

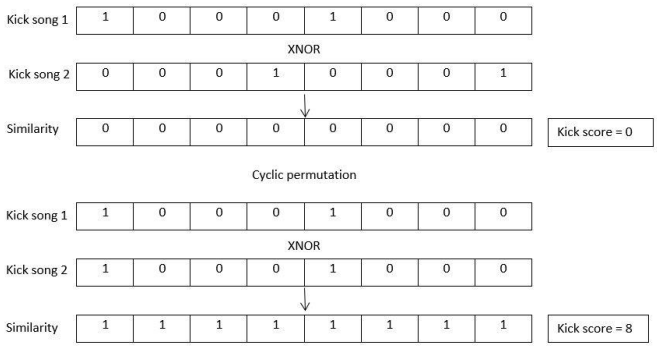

Fig 4: score computation between the kick events of 2 songs

The sum of the resulting 1's in the kick and snare arrays after applying the XNOR operator gives the kick and snares scores. These scores are then averaged using a quadratic mean.

$$score_{tot} = \sqrt{\frac{score_{kick}{}^2 + score_{snare}{}^2}{2}}$$

The advantage of using quadratic mean is that great values have a strong influence. So, if 2 songs have very similar kick content but different snare content, the score will be high. This is important because some specific hip hop subgenres can be detected just by looking at the kick pattern (e.g. "jersey") or the snare pattern (e.g. certain "drill" songs).

V. Song Combination

The goal of the song combination step is to find the combination of $N \leq n$ songs (where n is the number of songs recommended by a recommendation algorithm) among all the songs that have the higher score. This score is computed by combining the drum similarity score with a certain constant added if 2 songs have the same mode. However, enumerating all the possible combinations is not feasible in reality because of the exponential number of ways to combine the songs. One way to solve this problem is to introduce constraints.

The constraints will be that two consecutive songs can only differ by some pre-defined key and tempo values. The reason behind this choice is that in general, DJs tend to choose songs in their mix in a way that 2 consecutive songs won't differ by more than 1 semitone and 10% of tempo. We will keep these values as our constraints.

Then, it is possible to enumerate all the possible combinations given our constraints. To do so, we compute for each song all the songs that can follow it in the respect of the constraints. A recursive method is then used to compute all the possible combinations of N songs starting with each song. The algorithm is detailed in fig 5 where l is a temporary list that tracks each combination, $combinationList$ is the list with all the possible combinations and sl is a dictionary that maps each song with the song that can follow it in respect of the constraints.

If no combination is possible, we can lower the constraints and if N is very large, we can increase them. The score is computed for each possible combination and the combination with the greater score is selected as the optimal song combination.

```
Input: l (current song list), sl (song list), cs (current song), N
       (number of song for each combination)
Output: All the possible song combinations
Function songCombination(l, sl, cs, N):
    if l.length ≥N−1 then
        l.append(cs)
        global combinationList
        if l do not contains duplicates then
            combinationList.append(l)
        end
        return
    else
        l.append(cs)
        for n in sl[cs] do
            songCombination(l,sl,cs,N)
        end
    end
```

```
combinationList = [ ];
Function Main(l, sl, cs, N):
    for s in sl do
        songCombination(l,sl,cs,N)
    end
```

Fig 5: song combination algorithm

VI. Results and outlooks

Even if it is not possible to properly measure the quality of the overall song combination method, its results on real songs are very satisfying (despite the problems with snare detection). The song transitions are coherent, and the algorithm tends to group similar types of dance songs together. The computation time is also well reduced by the song combination algorithm.

as stated before, the purpose of this project is not to give an operational playlist ordering algorithm but to explore some methods that can help to order hip-hop dance playlists.

Some improvements could be brought to the method that was presented here. The first improvements concern drum detection. Apart from the fact that the dataset could be improved, this step is particularly computationally heavy because of the drum separation that takes at least tens of seconds to compute for each song. Onset correction is also not sufficient to ensure reliable drum detection when the drums are sophisticated. One way to solve this problem would be to use more complex models like convolutional neural networks that show good

performances on music processing tasks including sound event detection as suggested in the work of Gururani et al. on instrument activity detection [7]. Another way would be to train the model to detect drum onsets and not drum activity. The drum's subgenres could also be detected because some hip-hop sub-genres often have their specific type of kicks and snares. Some other music properties could be taken into account in the score computation like loudness and energy. These properties are important because automatically generated playlists often have brutal transitions between some consecutive songs (from a very calm song to a very loud or energic one and vice-versa).

This hip-hop dance song ordering method could be used to generate automatic mixtapes. This is a real need in the hip-hop dance community for the reasons developed in the introduction. Automatic mixtapes generation could also be useful for small parties where no DJ is available (to avoid interruptions at each song change and to have a coherent order in the songs). Finally, some elements of this method could also be used to order non-specific music playlists on streaming platforms.

VII. Conclusion

This project provides different tools and a more general method to optimally combine the songs of a given hip-hop dance playlist. The results are satisfactory on the whole, but some improvements could be made on drum detection.

References

[1] McFee, Brian, Colin Raffel, Dawen Liang, Daniel PW Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. "librosa: Audio and music signal analysis in python." In Proceedings of the 14th python in science conference, pp. 18-25. 2015

[2] Ellis, Daniel. (2007). Beat Tracking by Dynamic Programming. Journal of New Music Research. 36. 51-60. 10.1080/09298210701653344.

[3] Meinard Müller. (2015). FUNDAMENTALS OF MUSIC PROCESSING: audio, analysis, algorithms, applications. SPRINGER

[4] Bertin-Mahieux, Thierry & Ellis, Daniel & Whitman, Brian & Lamere, Paul. (2011). The Million Song Dataset.. Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011). 591-596.

[5] Mahieu Robert. (2016). "Detecting Musical Key with Supervised Learning

[6] Hennequin, Romain & Khlif, Anis & Voituret, Felix & Moussallam, Manuel. (2020). Spleeter: a fast and efficient music source separation tool with pre-trained models. Journal of Open Source Software. 5. 2154. 10.21105/joss.02154.

[7] Gururani, Siddharth & Summers, Cameron & Lerch, Alexander. (2019). INSTRUMENT ACTIVITY DETECTION IN POLYPHONIC MUSIC USING DEEP NEURAL NETWORKS.