AUTOMATED MUSIC TRACK GENERATION

LOUIS EUGENE
Stanford University
leugene@stanford.edu

GUILLAUME ROSTAING
Stanford University
rostaing@stanford.edu

***Abstract*: This paper aims at presenting our method to generate drum tracks to accompany guitar tracks. Our approach was to work on songs in the Midi format, separate them into several segments and then design our own features for the input guitar tracks as well as for the output drum tracks. We performed clustering algorithms (K-means, mixture of Gaussian, DBSCAN) as well as K-nn to group similar guitar tracks from the training set. Our main hypothesis was that guitar tracks that are close to each other might have close drum tracks as well. After assigning a test input to a cluster or set of neighbors, we generated a new drum track by randomly picking it in a specified group of existing drum tracks from the training set. This method has allowed us to obtain very good acoustic results.**

Introduction

Wouldn't the possibility to create a whole song from a single instrument track be terrific? If it is currently possible to add some computerized drum loops to accompany a guitar track, one can realize a good result is often hard to achieve due to the huge amount of different loops available, many of them being a poor fit to the original track.

Our goal in this project is, given a guitar track, to find the best possible drum track to accompany it. In order to cover the largest possible scope of situations, we are training our algorithms on a wide range of songs from different but close genres, mostly pop-rock/rock/metal songs. Our model is thus in its current state more intended to support applications in these musical genres.

1 Dataset

Representing music on a computer can be done in different ways. We personally decided to work on notational music, i.e. on the information contained in music scores. We built our own data set in the following way:

- We downloaded 200 free Guitar Pro tabs
- We worked on the tabs to remove every tracks except the guitar and drum tracks
- We exported the tabs in the Midi format
- We segmented the 200 songs in parts of exactly 16 beats

In the end, we obtained 5134 Midi files which constituted our data set. To read the midi format in Matlab, we used an existing library: Midi Toolbox (see [1]), which allowed us to represent our music segments by matrices (see 3-c).

2 Features

One of the most important part of the project was to design and choose our features. We tried to determine the most relevant features to perform our task by looking at the melodic and rhythmic behavior of our segments. Our goal was to find what features characterize the "style" of each music segment and which correlations exist between the different instruments (for example when the guitar plays a metal riff, drums are often fast with a lot of notes while when the guitar plays a solo, drums tend to fade).

a) Guitar features

We currently use the 16 following features:

- The number of notes
- The tempo
- The mean, max, min and standard deviation of the duration of the notes
- The mean, max, standard deviation of time intervals between two notes
- The number of different musical intervals, the mean, max and standard deviation of musical intervals
- The percentage of chords, number of different pitches and percentage of pitches different from the fundamental.

b) Drum features(target representation)

We aim at generating drum tracks. As a drum track is not a real mathematical value, we had to

engineer features like we did for the guitar track to work on it. We chose 11 features:

- The number of drum ticks ("notes")
- The tempo
- The mean, max, standard deviation of time intervals between two notes
- The number and proportion of appearance of pedal hi-hat, acoustic snare and acoustic bass drum which are the three staples elements amongst a drum track.

c) Extraction

To get the features from the Midi files we first used the function *readmidi_java* which loads the information contained in the Midi file in a n by 7 matrix with n the number of all the notes (a line is a note) and the column corresponding to the *onset time* (in beats), the *duration of the note* (in beats), the *midi channel* (integer between 1 and 16, traditionally 1 for the guitar and 10 for the drum), the *midi pitch* (the conversion of the pitch and octave in an integer (see [2]), the *velocity*, the *onset time* (in sec) and the *duration of the note* (in sec). Using the function *getmidich*, we split the matrix in two matrices corresponding to the two channels (guitar and drum). We worked on the separate matrices to obtain the features (from the guitar matrix) and the targets (from the drum matrix). We only extracted the features of the segments in which the two channels were present. Finally, we normalized the features between 0 and 1 in order to calculate norms with no flaws.

3 Model

a) Algorithms

The purpose of our work is to generate drums that can accompany guitars in an efficient way. Machine learning helps us go over thousands of famous songs to figure out what kind of patterns make different instruments fit together. Because music is a subjective field and there are no "labels" to describe music segments, we had to use unsupervised algortithms. We worked with: linear regression, K-means, K-nearest neighbors (K-nn), Mixture of Gaussians and DBSCAN.

Linear regression performed very poorly thus we abandoned it early. While K-means splits our data set in K clusters in which each observation belongs to the cluster with the nearest mean, K-nn gives for each point its K closest neighbors in the feature space. DBSCAN stands for "Density-Based Spatial Clustering of Applications with Noise". It is a data-clustering algorithm using the notion of density reachability. DBSCAN works with two parameters: a distance ε and a minimum number of points required to form a dense region (minPts). The principle is as follow: for each point of the dataset, we retrieve the ε-neighborhood (all points within distance ε) and check if the number of points is greater than minPts. If it is the case, we start a cluster and extend it to all the points ε-reachable from the cluster. Otherwise, we label as noise (the point might later be found in a sufficiently sized ε-environment of a different point and hence be made part of a cluster).

b) Music generation

We explored different technics to generate a drum track. Given a new input, we assigned its closest clusters in the feature (guitar) space using k-means, DBSCAN or Mixture of Gaussians or we fetched the nearest neighbors with K-nn. We then explored different ideas to actually output a drum track, based on the hypothesis that for a group of similar guitar tracks, the drum tracks should be close to each other as well:

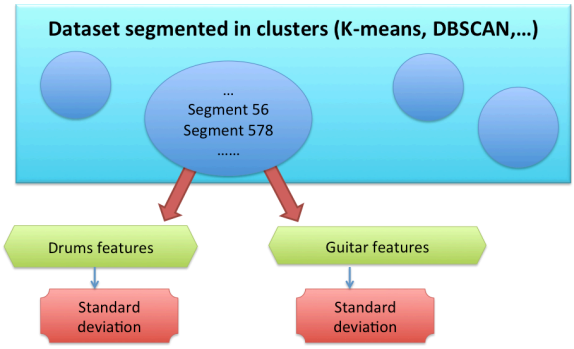
- We picked a drum track within the cluster or neighbors and assigned it to the test vector
- We performed K-means in the target (drum) space on the training set, determining the major drum cluster within a guitar cluster or set of neighbors (correlation) and then picking up a random drum in the drum cluster
- Based on the mean drum values within the cluster/set of neighbors, we can also think of randomly generating a track, which satisfy theses values. However we did not implement this method as we think it is hard to generate a computerized battery track and more research is required to build from scratch a drum track which sounds as good as a real track.

We then merged the two notes matrix in a new Midi file. Processing this way, we get a new segment ready to be played in a music player.

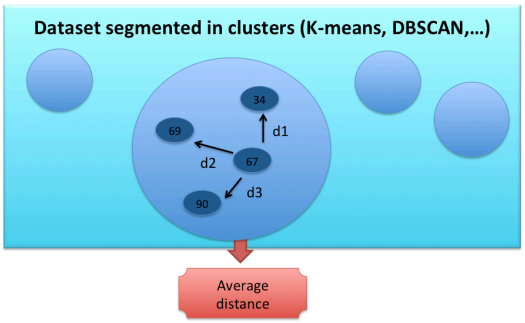
4 Results

Because we don't have any labels, it is hard to figure out how good our algorithms and choices of features are. To measure the relevance of our different clusters/set, we implemented our own metrics:

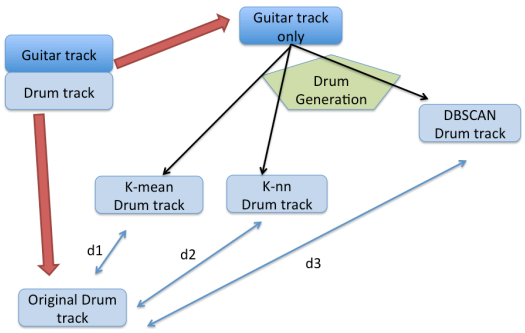
Metric 1: We decided to implement a function giving the standard deviation of the values of the features in each cluster/set of neighbors. We then compared these values to the standard deviations of the entire training set to get percentages (100% if equal). It works both for guitar features and drums features.



Metric 2: For each elements, we calculated the average Euclidian distances in the feature space between our segment and every other segments in its the cluster/set.



Metric 3: To have a sense of how "good" our drums generation are, we computed the Euclidian distance in the drum space between the generated drums and the original drums.



a) Features validation

To check the relevance of the features and to sort them by order of importance, we used our Metric 1 (standard deviations) for each algorithm in the different clusters/set generated. In other words, we tried to figure out which features were more homogeneous in the clusters/set and thus more characteristics of a cluster. To get an insight, we preceded using backward feature selection: we removed one feature at a time, computed our different algorithms and compared the standards deviation of the guitar features in the different clusters/set with n features and n-1 features. A percentage over 100% means that the feature is relevant (removing it resulted in an increased dispersion).

	K-means	DBSCAN	K-nn
Feature 1	104.13	144.53	98.0664158
Feature 2	104.02	103.07	104.206931
Feature 3	99.49	109.88	104.092311
Feature 4	103.30	126.23	103.395423
Feature 5	102.27	103.62	105.194378
Feature 6	98.07	169.44	99.8699416
Feature 7	100.25	124.23	98.0032114
Feature 8	100.34	119.29	93.9042858
Feature 9	99.07	144.73	111.343798
Feature 10	99.66	100.00	104.269363
Feature 11	101.42	100.00	104.461225
Feature 12	102.49	122.31	101.738106
Feature 13	97.58	228.44	376.031849
Feature 14	93.19	182.52	64.7508011
Feature 15	102.77	165.81	96.4470873
Feature 16	96.16	205.74	91.2584146

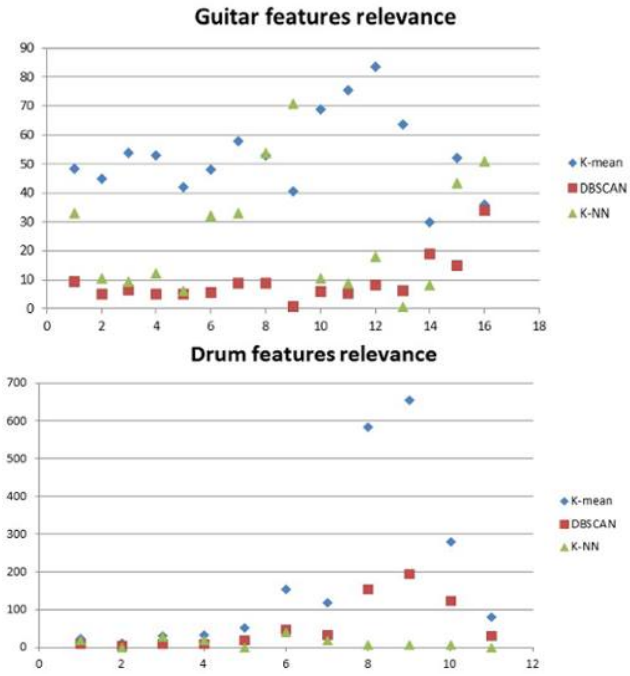
The guitar features are respectively: *number of notes* (1), *note duration mean* (2), *duration standard deviation (std)* (3), *duration max* (4), *duration min* (5), *number of different intervals* (6), *mean of the intervals* (7), *intervals std* (8), *interval max* (9), *time interval between 2 notes ("time") mean* (10), *time std* (11), *time max* (12), *tempo* (13), *chords percentage* (14), *number of different pitches* (15), *percentage of pitches different from the fundamental* (16).

In addition to the standard deviations of the different segments in each clusters/set in the guitar feature space, we also calculated the standard deviations of the different segments in the drum feature space.

In the drum space, the features are respectively: *number of ticks* (1), *proportion of hi-hat* (2), *proportion of acoustic snare* (3), *proportion of acoustic bass drum* (4), *number of hit-hat* (5), *number of acoustic snare* (6), *number of acoustic bass drum* (7), *time interval between 2 ticks ("time") mean* (8), *time standard deviation* (9), *time max* (10), *tempo* (11).

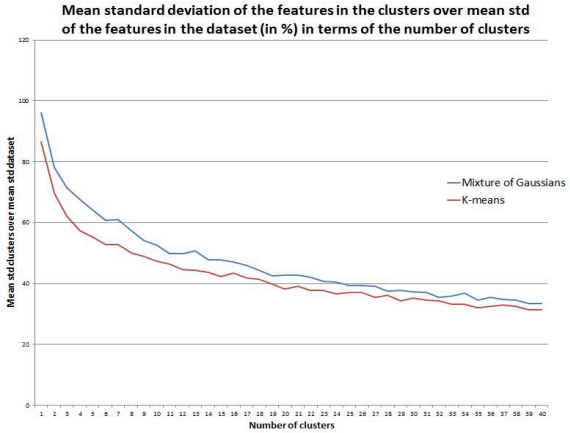
We also implemented backward feature selection with our Metric 3: we generated drums without one of the guitar features and computed the distances to the original drums.

Finally, we computed the mean standard deviation of all the features in the clusters/sets over the mean standard deviation in the whole training set for the 3 algorithms to know which features were more characteristic of a cluster/set and know which features were the best to discriminate between the segments.

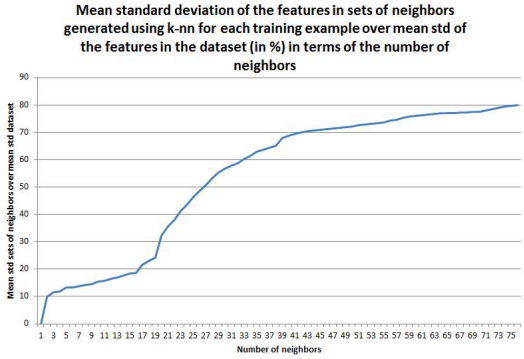


b) *Parameters selection*

In order to determine the right parameters to use in our algorithms, that is to say the number of clusters for K-means, the number of neighbors for K-nn, the minimal distance/number of points for DBSCAN, we trained our algorithms several times with different parameters and then computed, with method 1, the mean standard deviation of all the features in all the clusters/sets of points.



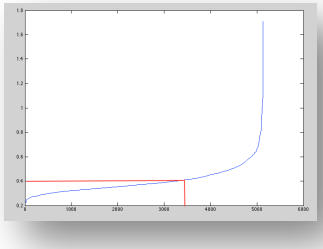
For K-means, as the number of clusters is increased, the deviation value decreases which is logical as clusters of fewer points are more representative of local structures. However, the computation time increases. We decided that a good trade-off was to select 50 clusters. We did the same for the mixture of Gaussian.



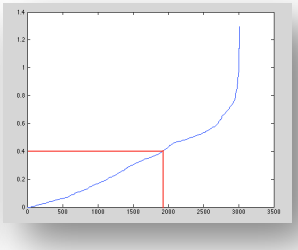
For K-nn, as the number of neighbors is increased the mean standard deviation increases (more dispersion) which is logical as we consider higher sets of points less representative of local structures. We determined that 20 neighbors was a reasonable choice. For DBSCAN, we ended up with a distance of 0.2 and a number of points minimum to form a cluster of 4.

Once we had a good choice of parameters, we used our Metric 2 to compare the relevance of the different algorithms: we calculated the distances between each point and their neighbors/other points of clusters, both in the guitar space and the drums space. The following graphs give these distances, ordered and with an indication at a distance of 0.4:

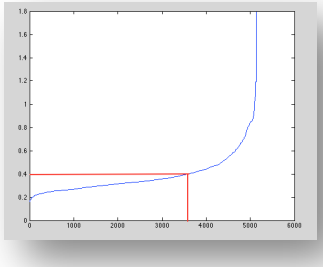
Guitars – Kmeans



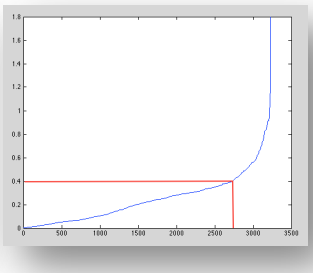
Guitars - DBSCAN



Drums – Kmeans



Drums – DBSCAN

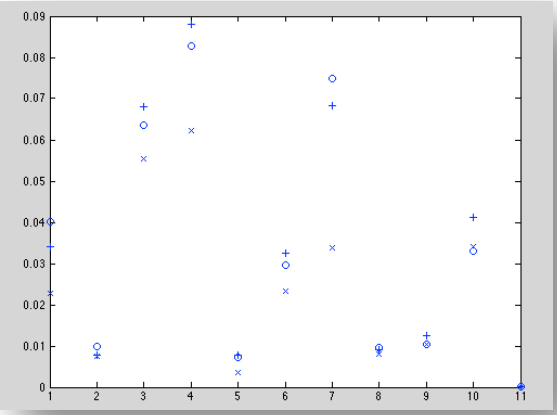


c) Drum generation results

We ran through our entire dataset and generated multiple times different drums for each segment and each algorithm. We then used our Metric 3 to compare the distances between the generated drums and original drums. We found:

DBSCAN (x): 0.1251
KMEAN (o): 0.1825
KNN (+): 0.1885

Here is the detail, feature by feature:



5 Conclusion

The results presented in the previous section have been mostly used to discriminate between the different algorithms but do not and cannot really provide a deep insight in the actual result, i.e. the acoustic output of a generated track. In some cases, the drum generated is pretty close to the original drum and the result is excellent to hear. But sometimes, the drum sounds quite differently and yet, the acoustic output is also very good even if it differs from the original song. In fact, trying to generate music is inherently subjective and it is extremely hard to create accurate metrics. That is why we mainly used ours to classify the algorithms and restrained ourselves from creating an absolute training and test error that we could have done by taking the mean of the targets values for a cluster or set of neighbors.

6 Future work

We currently generate drums for guitar segments of 16 beats. We already implemented a function to generate drums for an entire song by sub-segmenting the song, generating drums segment by segment and merging the results. However this doesn't give much consistency in the progression of drums. We can think of implementing hidden Markov models / Kalman filters to have a smooth transition of drums sequences (the goal is to estimate what the drums should be based on the previous drums segments and merge the result with the actual drum generation from our algorithms).

All of our work has been focused on generating drums for guitar segments but we would like to adapt our algorithms to other combination of tracks: bass from guitar, guitar from drums, etc…

We currently use midi files for our algorithms. Our goal is to be able to generate drums based on a real guitar recording. We started to implement an algorithm able to convert a .wav into a .mid but the results are imperfect for now: our algorithm is able to efficiently recover the general pitch of a sound but we are still unable to tell if it is a chord or a single note and we have trouble in finding the correct starting time and ending time of the different notes. Because our features depend heavily on the times intervals, our generation of drums from .wav are still imperfect.

Finally, if we can solve the previous problems, we would like to write our algorithms in Objective C and Java in order to make a mobile application offer our work to the world and make musicians happy.

REFERENCES

[1] EEROLA Tuomas and TOIVIAINEN Petri, MIDI Toolbox: MATLAB Tools for Music Research, University of Jyväskylä, www.jyu.fi/musica/miditoolbox/, 2004.

[2] http://www.tonalsoft.com/pub/news/pitch-bend.aspx