Automated DJ: MLSP Fall 2021

Michael Li, Sasikala Mani, Yiren Zhou
Carnegie Mellon University
mcli@andrew.cmu.edu, sasikalm@andrew.cmu.edu, yirenzho@andrew.cmu.edu

1 Preliminaries

1.1 Introduction

As a Disk Jockey (DJ), seamless transitions between music tracks are crucial in maintaining audience engagement and excitement by creating unique combinations of songs. Our goal is to find potential transitions between tracks to automate DJing. Such research and technologies are important for understanding musical cultures and receptions, as well as exploring the possibility of improving and automating music technology in the industry of entertainment. The scope of our project is to focus on electronic dance/house music, which is the most common form of music used by DJs in the entertainment industry.

1.2 Related Work

There are many ongoing efforts to create automated cue-point findings. MSAF [1] is an open-source Python framework that has a variety of algorithms that find potential musical boundaries/cue-points. The API exposed by MSAF allows us to specify the algorithm and the feature we want to use. Algorithms include Checkerboard Kernel, Convex Non-Negative Matrix Factorization, and Variable Markov Oracle, etc. Available features include MFCC, CQT, and tonal centroids, etc. and are mostly implemented using librosa [2]. Our approach has been based on the algorithms provided by the MSAF library.

Zehren et al. [3], proposes an algorithm for finding cue points in EDM tracks using a variety of feature extraction and deep-learning-based rhythm detection algorithms together to find cue points.

1.3 Dataset

We used the M-DJCUE dataset [4], created by the authors of [3], which contains annotations and cue points for 134 EDM tracks created between 1987-2016. These cue points are picked by experts, and allow for us to have a sense of "ground truth" when evaluating our algorithm.

This section discusses the format of our dataset which was pre-processed for our Adaboost algorithm, which will be elaborated on in Section 1.4.

Our dataset was a directory of ".jams" files that contained the YouTube links to the songs as well as the cue points picked out by experts. We pre-processed these files so that the input was ".webm" files that we can load as audio data. The output was a list of cue points or boundaries where each cue point was represented as a timestamp in the format "min:sec".

Since each ".jams" file has multiple annotators with their musical boundaries, we sorted and then cleaned the raw data by grouping all boundaries within 1 second of each other together, as they likely refer to the same boundary. We then formatted this output (a list of cue points in seconds) to follow the same format as the experiment (i.e., min:sec).

There is one other step in pre-processing our ground-truth data, which is to convert all cue-points into a row vector in an n-dimensional space, where n is the number of algorithms from the MSAF

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

[1] library used to extract the cue-points. For example, with $n = 2$ (i.e., two algorithms), given a cue-point x, if the first algorithm outputted x as a cue-point while the second did not, the resulting row vector will be $[1\ 0]$, where the 1 at index 1 denotes that the first algorithm did output the cue point and the 0 at index 2 denotes that the second algorithm did not. More details will be discussed in the next section.

2 Initial Designs

2.1 Non-Working Experiments

We have experimented with two different approaches that did not work well: the Variable Markov Oracle (VMO) algorithm, and the Non-negative Matrix Factorization. Both of them will be explained below.

For the NMF approach [6], the main idea is two identify shifts from one prominent frequency to another, which could be potential cue-points. In music, most transitions happen when an instrument fades out and another instrument comes in. Because different instruments operate at different frequencies, these "shifts" can be found by extracting the individual "bases" using audio separation and looking for outstanding changes in their magnitudes.

We used the code from an open-source project [7], which implements a collection of audio separation tools based on various papers. We performed NMF on songs from the same M-DJCUE dataset, plotted the wave plots of the bases, and extracted the "cue-points" from outstanding changes in their magnitudes. For example, for the song "Tuna Melt", a song in M-DJCUE which consists of rather a small set of instruments, the wave plots of the three separated bases are in Figure 1:

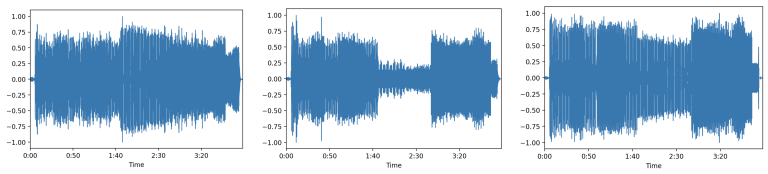

Figure 1: Wave plots of the three estimated bases.

However, we realized that using NMF is impractical for the task because the usage of instruments varies significantly in different pieces of EDM music. Hence, it is difficult to generalize how many features we want to extract given a set of songs. In the above example, the instruments are not well separated as the bases all follow a similar pattern.

We also experimented with algorithms provided by the MSAF library [1], as they are also used in [2] to find cue points. When evaluating the Variable Markov Oracle (VMO) [5] algorithm in MSAF, we found that it was unsuccessful at effectively finding cue points in EDM music.

VMO uses a form of pattern matching to find musical boundaries. Since EDM music is highly repetitive, VMO will return a dense output of matches. This leads to high recall but very low precision, as VMO will output the correct boundaries and many false positives.

We tested VMO using songs and ground truth annotations from the M-DJCUE dataset [3] in it, we found that VMO had a very low precision of 0.007444168734491315, a decent recall of 0.6, and an F-score of 0.014705882352941178.

2.2 Working Experiment: MSAF Majority Vote Algorithm

A working experiment we conducted was to create a basic "Majority-Vote" algorithm with the six algorithms supplied by MSAF [1]. Some of the MSAF algorithms have high recall but low precision (like VMO), and some are the opposite - they have high precision and low recall.

A basic "majority-vote" algorithm using the MSAF boundary algorithms runs every algorithm, and only returns boundaries that the majority of algorithms output. This way, it might be able to return musical boundaries with higher precision.

To illustrate, the plot below compares the majority-vote's returned boundaries, VMO's returned boundaries, and the ground truth for "Tuna Melt", a song in M-DJCUE. A vertical red line represents the timestamp of a musical boundary.

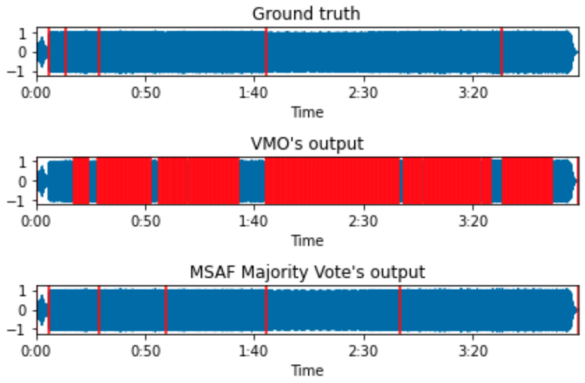

Figure 2: A comparison of musical boundaries from VMO, the majority-vote and the ground truth.

We found that the majority-vote had a precision of 0.42857, the same recall of 0.6, and an F-score of 0.5. This is a vast improvement on VMO's precision, while maintaining the same recall.

Comparing this to published results on experiments with MSAF, we find similar results. Zehren et al. [3][4] tested the MSAF algorithms on the M-DJCUE dataset and found that the best-performing algorithm, OLDA, yielded a precision of 0.3 (they didn't list the recall). Based on their examination of MSAF, we find that the majority vote algorithm gets slightly improved results in the same realm as the highest-performing algorithm in MSAF.

Overall, this experiment demonstrates that combining the MSAF algorithms as an ensemble may yield more robust cue points for DJing than using any of the MSAF algorithms alone. Hence, we think that it is representative of an upper bound of the "performance" of any MSAF algorithm. It will be used as our baseline to compare the MSAF algorithms to our new algorithm. We plan to design more sophisticated ensemble methods with these algorithms, elaborated in the next section.

3 Final Design: MSAF-Adaboost

3.1 Training Adaboost on top of MSAF's Outputs

In Section 1.2, we introduced the MSAF library [1] which contains a number of algorithms for extracting cue points from a music file. Our final approach has been built on top of the library. In this section, we will elaborate on our approach to identifying cue points by combining MSAF algorithms and Adaboost [8], which we named the "MSAF-Adaboost" Model. We chose Adaboost because it was the first classifier that we learned in class, and it can learn a complex decision boundary using its weak classifiers.

The overall goal of using Adaboost on top of the MSAF algorithms is to filter out false positives and increase precision. We can create a dataset with the cue points that MSAF outputs and labels for whether those cue points are in the ground truth, and train Adaboost models on this dataset to differentiate between correct and incorrect predicted cue points.

Suppose that there are x songs in M-DJCUE and n algorithms in the MSAF library [1]. This model will train x Adaboost models, one for each song. For each song i where $i \in [1, x]$, we run each MSAF algorithm j where $j \in [1, n]$ and retrieve the cue points for this song. Some algorithms could output the same set of cue points. Hence, we could model each cue point in an n-dimensional space, where a cue point is $\{0, 1\}^n$. Each coordinate represents if the cue point was predicted by a particular MSAF algorithm. In other words, each of the cue points is transformed into a point $(c_1, c_2, ...c_n) \in \{0, 1\}^n$, where each coordinate c_i, $\forall i \in [1, n]$, is 1 if the i-th algorithm outputted the cue point, or 0 otherwise.

Given this dataset where each row is a cue point represented in an n-dimensional space (again, n as the number of MSAF algorithms), we can create a classification label for each row/cue point with

the ".jams" files in the M-DJCUE dataset. The ".jam" files, which contain DJ-picked cue points (timestamps) for all songs in the M-DJCUE data set, can serve as the "ground truth" labels for classifying each cue point.

Based on whether each cue point in the dataset is actually in the ground truth, we create a column vector of labels $[l_1, l_2, ...l_n]^T$ where each label l_i is 1 if the cue point represented by row i in the dataset is in the ground truth, and 0 if it isn't.

Based on the dataset and labels, we can train an Adaboost model for a specific song to differentiate between correct and incorrect predictions from the MSAF algorithms and reduce false positives.

The above workflow could be visualized using Figure 3 below.

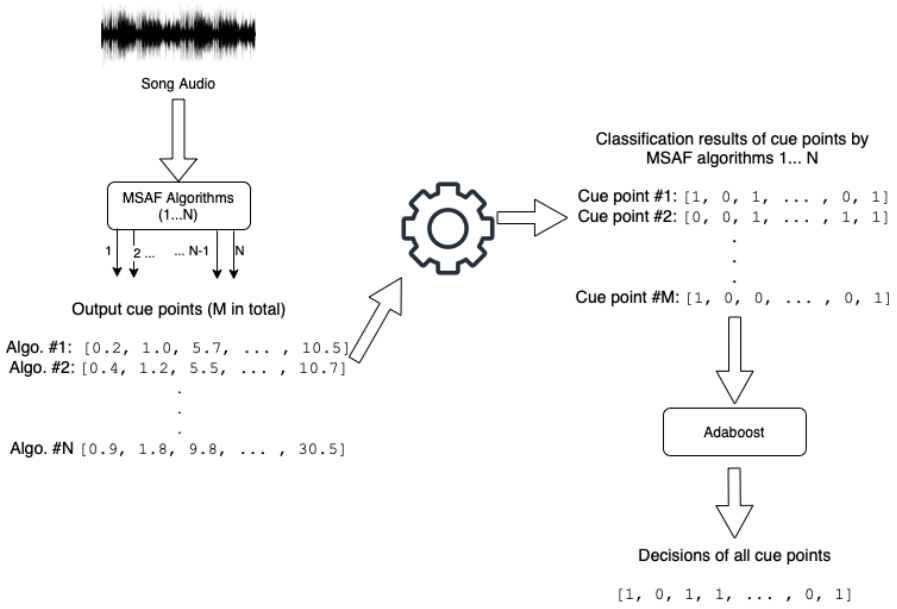

Figure 3: Pipeline for training cue points.

3.2 Making Predictions for New Songs the Trained Adaboost Models

With the trained models using Adaboost from the last section, for a new song, we ran the same set of n algorithms of MSAF to extract all candidate cue-points of the song. We then encoded each cue-point into a row vector in the n-dimensional space in the same approach as in our training phase.

Then, we had each of our Adaboost models predict the labels for each row vector (i.e., each cue-point). The labels would correspond to if the Adaboost model believes that the cue point is in the ground truth or not.

The final step was to return only the cue-points voted by the majority (over $\frac{1}{3}$ of the models) of the Adaboost models. Also, cue-points that were apart by 5 seconds were merged into a single cue-point because they were likely referring to the same moment of the song.

4 Results

4.1 Experimental Results

As mentioned in our midterm report, we have used precision and recall to evaluate our approach. Precision is the ratio of the number of correctly predicted cue-points over the total number of candidate cue-points outputted by our approach; Recall is the ratio of the number of correctly predicted cue-points over the total number of cue-points from the ground truth.

The evaluation metrics depend on two variables: the number of songs used in the training stage, and the number of weak classifiers used in Adaboost. Table 1 shows the values of precision and recalls with varying numbers of songs and weak classifiers used.

$(Num_{songs}, Num_{classifiers})$	Precision	Recall
(10, 50)	0.3876	0.3434
(10, 100)	0.3959	0.3452
(20, 20)	0.2937	0.4306
(40, 200)	0.3360	0.4520
(72, 10)	0.3815	0.3897
(72, 300)	0.3640	0.4289

Table 1: Precision and recall values based on varying numbers of songs and weak classifiers.

In comparison, the MSAF-based Majority Vote algorithm (the working experiment in the midterm report) had a precision of 0.2742123687281214 and a recall of 0.4181494661921708 when evaluated across the entire M-DJCUE dataset.

Our literature search found that other researchers working with MSAF had similar performance when experimenting with MSAF's algorithms for musical boundary detection. Zehren et al. [2][3] also evaluated the MSAF algorithms on the M-DJCUE dataset and found that the best-performing algorithm, OLDA, yielded a precision of 0.3 (they didn't list the recall).

Figure 4 shows a scatterplot comparing the precision and recall of the two algorithms, with each MSAF-Adaboost datapoint label representing $(Num_{songs}, Num_{classifiers})$:

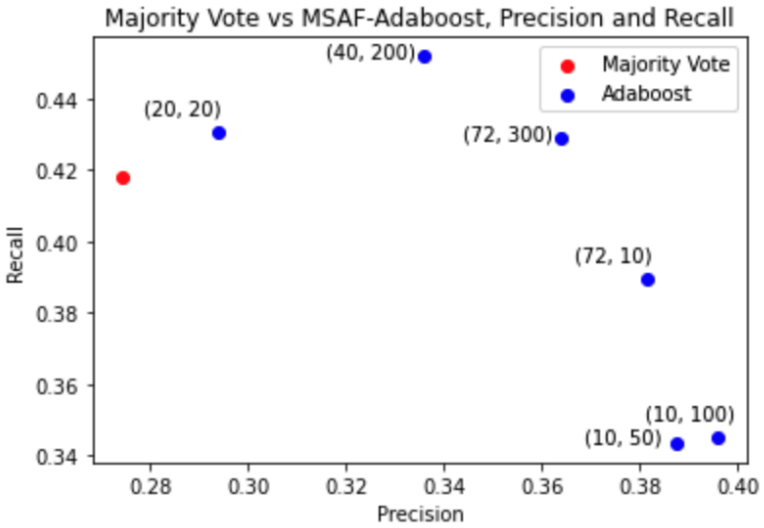

Figure 4: Precision and recall values for Majority Vote and MSAF-Adaboost over varying parameters.

4.2 Discussion and Analysis

As shown by the results, there is a general correlation between the number of songs used to train the model and the model's resulting recall. One possible explanation for this correlation is that, as more Adaboost models are trained on different EDM songs in the dataset, the majority of models will "learn" more intrinsic attributes within the unfiltered cue points that are only in correct cue points, and allow the overall model to return more correct cue points in its predictions.

Compared to the Majority Vote algorithm, the MSAF-Adaboost model performs better in terms of both precision and recall. Given that the Adaboost models for each song are trained to distinguish between ground truth cue points and false positives returned by the MSAF algorithms, the Adaboost-based model is expected to reduce false positives and get higher precision than Majority Vote. This

is reflected in Table 1 and Figure 2: compared to the majority vote algorithm, the Adaboost-based model gets consistently higher precision across all parameters we tested and both higher precision and higher recall in the cases of parameters $(20, 20)$, $(40, 200)$ and $(72, 300)$.

References

[1] Nieto, O., & Bello, J. (2016). Systematic Exploration of Computational Music Structure Research. In Proceedings of the 17th International Society for Music Information Retrieval Conference (ISMIR).

[2] Brian McFee, Colin Raffel, Dawen Liang, Daniel Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. librosa: Audio and Music Signal Analysis in Python. In SciPy, pages 18–24, Austin, Texas, 2015.

[3] Zehren, M., Alunno, M., & Bientinesi, P. (2020). Automatic Detection of Cue Points for DJ Mixing. arXiv [cs.SD]. Opgehaal van http://arxiv.org/abs/2007.08411

[4] Zehren, M., & Alunno, M. (2019). M-DJCUE: A MANUALLY ANNOTATED DATASET OF CUE POINTS.

[5] C. Wang and S. Dubnov, "Variable Markov Oracle: A Novel Sequential Data Points Clustering Algorithm with Application to 3D Gesture Query-Matching," 2014 IEEE International Symposium on Multimedia, 2014, pp. 215-222, doi: 10.1109/ISM.2014.39.

[6] Lee, D., & Seung, H. (2000). Algorithms for Non-Negative Matrix Factorization. In Proceedings of the 13th International Conference on Neural Information Processing Systems (pp. 535–541). MIT Press.

[7] Audio Source Separation (2020), GitHub Repository, https://github.com/tky823/audio_source_separation

[8] Freund, Y., & Schapire, R.E. (1999). A Short Introduction to Boosting.