GENERATING ANNOTATED TRAINING DATA FOR DRUM DETECTION IN POLYPHONIC AUDIO

Jarred Hawkins
University of Victoria
jarred@uvic.ca

Shae Brown
University of Victoria
shaeb@uvic.ca

ABSTRACT

This document describes a software tool to generate ground truth drum event annotations from polyphonic audio. We aim to reduce the time it takes to gather ground truth drum event annotations. The tool will estimate drum events in polyphonic audio using a trained model and then allow for manual correction. The application can be used as a semi-automated tool for generating more data in order to improve its own model further and generate data for future research. By building a visual tool, we can visually analyze the accuracy of the predicted drum events, manually fix the errors, and use the new ground truth to retrain the model. We implement a known method called “segment and classify” to predict percussive audio events. Our completed tool is open sourced. No quantitative results about work speed increases are presented in this paper.

1. INTRODUCTION

In many music genres, researchers such as Tanghe in [1] have noted that rhythmic structures have become at least equally as important as melodic or tonal structures. Percussive structures can provide valuable insight into a music piece. In order to use machine learning for classifying drum events, models are usually trained using audio and ground truths of the percussion events in this audio. According to Tanghe in [2], there are two main methods of generating training data. The first method uses a MIDI drum track and synthesizes an output. This method results in accurate tags, but with an unrealistic audio track. The second method is to manually tag the polyphonic audio. This manual tagging is tedious and slow, and as Tanghe noted, even experienced percussionists have difficulty annotating certain aspects (such as fast rolls, or brush strokes). Copyright restrictions prevent a centralized database of ‘real audio’ transcriptions from being feasible, meaning that finding quality annotated ground truth data of percussion events is difficult and required for each new application. In our work, we seek to provide a model-assisted method to ease and expedite the generation of quality ground truth training data for the purpose of drum detection in polyphonic audio.

2. RELATED WORK

2.1 Annotating Drum Event Data

Previous attempts to improve the process of annotating drum event data exist. Most notably in [1], Tanghe notes the difficulties of labeling fast drum notes (ie drum rolls) and issues distributing the music with the annotation data due to copyright restrictions. Tanghe also noted that visualizing events and the audio at the same time, together with the hearing the annotated events proved to be essential for the annotators. In implementing our tool, these findings were considered. Our attempts at locating large data sets proved unsuccessful, largely due to the mentioned copyright restrictions in redistributing music.

2.1 Drum Detection in Polyphonic Audio

There have been many approaches to detect drums in polyphonic audio. These range in accuracies and the types of data. In [3], Paulus and Virtanen achieved a 96% hit accuracy in detecting drum events by using non-negative spectrogram factorization in drum-only audio. In [2], Tanghe reached roughly 61% accurate results of detection on polyphonic audio. From these results we believe that a reasonable (able to be easily human-corrected) initial estimate is feasible. We expect that the human-time spent annotating could be reduced.

3. APPROACH

In this project, we seeked to train a classification model using Support Vector Machines (SVMs). Using a trained model, we will implement and run a modified version of the algorithm proposed by Tanghe in [2]. This algorithm is run on user input audio. From the output of this we will have an estimation of annotated drum events, which a user will manually modify and ensure the correctness of. This modification will take place in a web interface. Once a user modifies the drum events, they can export the annotated audio with drum annotations. This exported data can be used to retrain the classification model resulting in closer estimations for future iterations of the annotation step.

Several known approaches exist for solving the drum transcription problem, and most approaches can be categorized as segment and classify or separate and detect. [4]. Our proposed approach falls into segment and classify. This method can be described by identifying segments of significant events, and attempting to extract meaningful features of about events in order to classify them. The process can be separated into three parts: onset detection, feature extraction, and model training.

3.1 Onset Detection

The largest known weakness for the segment and classify approach is the possibility of undetected events at the segment phase [4]. Our onset detection aims to identify false positives rather than false negatives so that no potential drum events are lost at this stage.

3.2 Feature Extraction

Using the potential drum events found in the onset detection phase, we can extract features from the audio segment occurring after the event. Tanghe's paper [2] gives valuable insight to features relevant for classification: zero-crossing rate, spectral centroid, RMS (with and without filters), crest factor, spectral kurtosis, spectral rolloff, spectral skewness, spectral flatness, Mel frequency cepstral coefficients, and spectral bandwidth. We used each of these features in our approach.

3.3 Model Training

Initial training data was gathered from the DREANNS project [6] and the MDB Drums data set [11]. These training sets contained samples of 41 percussion annotated audio tracks, of varying length. As noted in the introduction, annotating percussion tracks is a painstakingly slow and difficult process, so we elected to not annotate any data ourselves.

Our initial model used an SVM classifier. According to Tanghe in [2], a SVM "is a data driven method that constructs a classifier that separates the data in a training set with low error and large margin". For a given audio file, we constructed a vector of audio features for each audio segment following a detected onset. A vector of ground truth values to classify each onset event as a percussion type or non-percussion can be constructed with the given annotated files. Using the scikit-learn library [7], a SVM can then be fitted to the provided data. Using the visualization tool, we can analyze the accuracy of the model's predictions with new data, and manually fix errors to create more training data.

4. IMPLEMENTATION

In this section we discuss the actual implementation of the software we created. Following in Figures 1 & 2 are diagrams of the software interaction. The user-level interaction will take place in a web browser. A web browser was chosen due to it's cross platform support, and the time savings with not having to learn a language specific GUI framework. We chose to implement the back end of the project in Python 3. To achieve this we used Flask, a Python web framework. Librosa and scikit-learn were two libraries that we utilized heavily for the audio feature extraction and model training, respectively [10, 7].

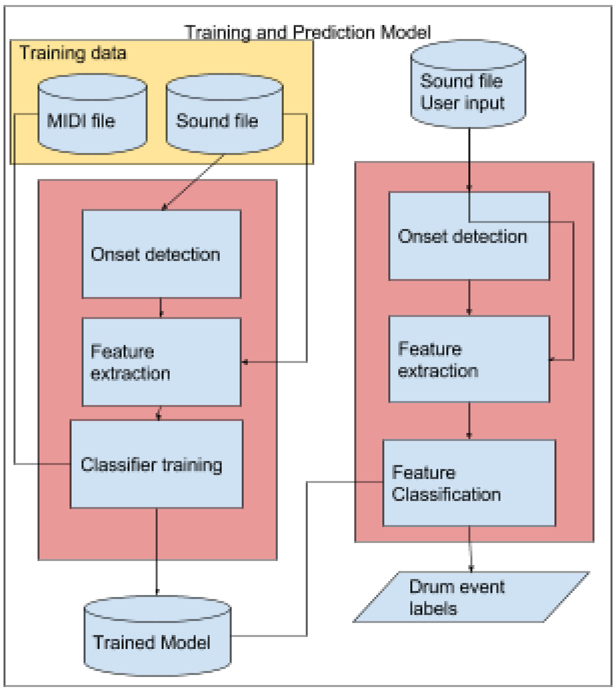

Figure 1. Diagram of application and training procedure of drum detection. Adapted from algorithm described in [2]

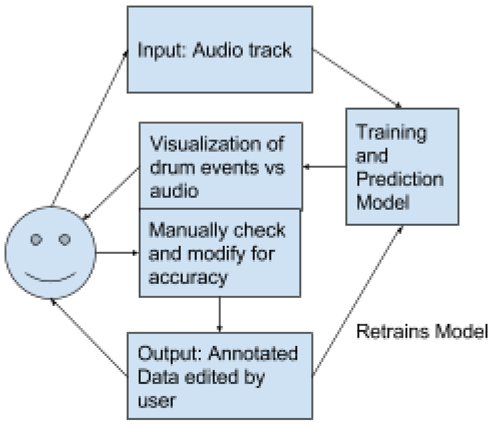
Input: Audio track
Training and Prediction Model
Visualization of drum events vs audio
Manually check and modify for accuracy
Output: Annotated Data edited by user
Retrains Model

Figure 2. Diagram of User-level interaction with the software

4.1 Front End

The front end has two screens: the initial upload screen, and the annotation screen. When a user opens the page, they are taken to the upload screen where they can upload a file of their choosing (.wav and .mp3 are currently supported). Once this file is uploaded it redirects the user to the annotation page. The initial annotation is processed by the server.

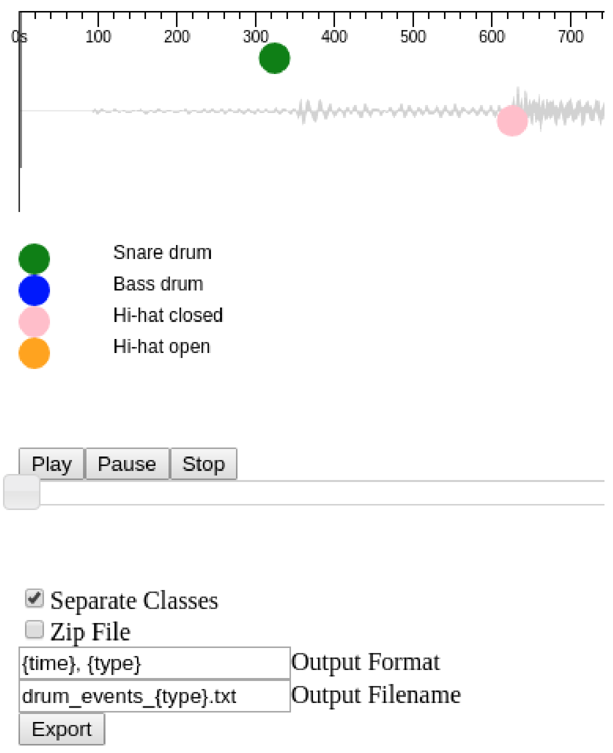

Figure 3. Annotation page of the front-end interface.

In *figure 3*, the main annotation screen is shown. The visualization in implemented using D3.js [8]. Controls on the annotation allow the user to control playback of the music, and export the annotated events. These annotated events are represented as a series of circles overlaid on the audio waveform. The annotated events can be adjusted by dragging them left or right, or reclassified by right clicking on the circle. The right click menu that pops up allows the circle to be removed, or traded for another class. Missing drum events can be added by double clicking where they occur. This is all computed client-side in the browser.

4.2 Back End

The web backend is implemented in Flask [9] and provides the front end with the times of each type of drum event. The flow for these events is as shown in Figure 1. Our initial approaches lost over half of the drum events, even at a low time precision (rounded to the nearest second). We originally used a class set of 16 drum types. However, for less prominent and less frequently occurring events the prediction accuracy, precision and recall was low. Our initial performance for the bass and snare drum were acceptable.

To remedy this, we performed several steps: changed the precision of the onset detection, cut the feature classes down to 4 classes, extracted more features from the audio, included more training data, and exported the extracted features and ground truth values from our training data into a pickle files for faster training.

Our initial onset detection results were invalid due to the precision we used in detecting each onset. Because we were using a precision of 500ms, any drum event within 500ms of a ground truth event would be classified as a true positive in our classifier. We have chosen 100ms precision as a nice balance between allowing small error in our onset detection, and unintentionally allowing multiple true positives for one ground truth event. From testing using our front-end, it seems to rarely occur where more than one of the same drum event are within 100ms of each other. Using this precision, approximately 70% of training data events are detected at onset detection phase.

We decided to cut our class set down to 4 classes. The classes we elected to keep were: Bass drum, Snare drum, Open hi-hat, and Closed hi-hat. These were chosen to their frequency of use in Western popular music. Cutting down the class set provided us with improved results as shown in Table 1.

Class	Accuracy	Precision	Recall
Bass drum	0.625	0.345	0.347
Hi-hat closed	0.589	0.35	0.35
Hi-hat open	0.898	0.035	0.051
Snare drum	0.567	0.414	0.414

Table 1. Results after shrinking class set.

We still were not satisfied with these results, so we seeked to extract more features from our audio files. See Table 3 for the newly selected features. By doing more research, we discovered a new set of publicly available training data called MDB Drums [11]. We added this to our training set. With these new features and training data added, our results improved by a 5% average across each class.

Class	Accuracy	Precision	Recall
Bass drum	0.674	0.43	0.43
Hi-hat closed	0.628	0.409	0.404
Hi-hat open	0.89	0.071	0.082
Snare drum	0.598	0.456	0.456

Table 2. The results have increased by an average of 5%

after adding the new features.
We continued to experiment with the different scikit-learn classifiers which support multilabel classification. The decision tree continued outperform other classifiers. We elected to use a decision tree as opposed to the SVM that we initially proposed.

5. FEATURES

A positive in using a decision tree was the ability to determine the importance of each feature. The importance is "computed as the (normalized) total reduction of the criterion brought by that feature." [12]. This value, also known as the gini importance, is shown in column three below. The table is sorted by importance.

Feature name	Description	Importance
Spectral Flatness		0.103996
RMSb1	RMS of filter #1	0.07799
Spectral Bandwith*		0.074742
Crest Factor*		0.074696
RMSb2	RMS of filter #2	0.072094
MFCC		0.058477
Spectral Rolloff		0.0581
RMSbRel23*	RMSb2 - RMSb3	0.055638
Spectral centroid		0.053431
Spectral Kurtosis		0.044224
RMSb3Rel*	RMS - RMSb3	0.044037
Zero crossing rate		0.043752
RMSbRel12*	RMSb1 - RMSb2	0.039394
Spectral Skewness		0.039224
RMSb2Rel*	RMS - RMSb2	0.03638
RMSb3	RMS of filter #3	0.034385
RMS	of the overall clip	0.029092
RMSbRel13*	RMSb1 - RMSb3	0.032416
RMSb1Rel*	RMS - RMSb1	0.027932

Table 3. Importance of each feature in decision tree ranking. The features designated with an asterisk were added after the presentation and contributed to the 5% increase in classification accuracy.

5.1 Filters

The features labeled RMSb1, RMSb2, RMSb3 refer to the RMS of three different frequency bands defined by Tanghe [2]. The three frequency bands are acquired using butterworth filters. Table 4 shows the values used to construct the three filters.

Passband Frequency	Stopband Frequency	Passband Ripple	Stopband Attenuation
[49, 50] / nyq	[0.01, 2000] / nyq	0.01	62
[200, 201] / nyq	[1, 1300] / nyq	0.01	20
[5100, 16300] / nyq	[65, 22000] / nyq	0.05	60

Table 4. Different filters used. *nyq* refers to the nyquist frequency.

5. CONCLUSION

5.1 Conclusion

In this paper we described a framework to generate annotated training data of percussion events in polyphonic audio. This system involves a human in the loop to verify and improve the prediction model's accuracy. We also provided an example implementation, taking into consideration previous efforts at annotating percussion events. In anecdotal experimentation our framework is effective and intuitive to a human annotator. We faced some difficulties addressing many classes of percussion events, so we restricted our class set to bass drums, snare drums, and open and closed hi-hats. This improved our results significantly.

5.2 Future Work

Our work seems to be intuitive and offer speed improvements, but we do not provide any quantitative measures about annotation speed improvements. In the future a study examining the effectiveness of this framework would be helpful. Additionally, we were not able to achieve as high of an event classification accuracy as we would have liked. In the future improving this accuracy and re-introducing more classes of drum events would benefit the effectiveness of this framework. Another approach to look into is using a small subset of training data for a set of similar audio files and evaluating the classifiers performance on the similar audio files. This would allow faster iterations on similar themed sets of audio (at the cost of a less-generalized model). Additional remaining work includes automating retraining the model when new data is annotated in the UI. However, this comes at the cost of users possibly training the model with poorly annotated files. At this time, the model can be manually retrained with new data by exporting the annotations through the UI, placing the files in the correct directory, and following the instructions in the readme.

5.3 Demo and Code

Upon completion of the project, we open sourced our code. The source code is available at:
https://github.com/ShaeBrown/csc475.
A demo is available at:
https://drum-annotation.herokuapp.com/.
Note that mp3 files do not work on our heroku deployment at this time.

6. CITATIONS

[1] Tanghe, K., Lesaffre, M., Degroeve, S., Leman, M., De Baets, B. and Martens, J. (2005). Collecting Ground Truth Annotations for Drum Detection in Polyphonic Music.

[2] Tanghe, K., Degroeve UGent, S. and De Baets, B. (2005). An algorithm for detecting and labeling drum events in polyphonic music. *Proceedings of the first Music Information Retrieval Evaluation eXchange (MIREX)*.

[3] Paulus, J. and Virtanen, T. (2005). Drum Transcription with Non-Negative Spectrogram Factorisation. Signal Processing Conference, 2005 13th European.

[4] Paulus, J. and Klapuri, A. (2009). Drum Sound Detection in Polyphonic Music with Hidden Markov Models. EURASIP Journal on Audio, Speech, and Music Processing, 2009.

[5] Paul Brossier, MartinHN, Eduard Müller, Nils Philippsen, Tres Seaver, Hannes Fritz, and Sam Alexander, “aubio/aubio: 0.4.6”. Zenodo, 04-Oct-2017.

[6] Ricard Marxer, Jordi Janer, "Study of Regularizations and Constraints in NMF-Based Drums Monaural Separation", Proc. of the 7th Int. Conference on Digital Audio Effects (DAFx’13). Maynooth,Ireland,2013. https://www.upf.edu/web/mtg/dreanss

[7] Pedregosa et al., "Scikit-learn: Machine Learning in Python", Journal of Machine Learning Research, vol. 12, pp. 2825-2830, 2011.

[8] http//d3js.org/, retrieved on November 6th, 2017

[9] http://flask.pocoo.org/, retrieved on November 6th, 2017

[10] Brian McFee, 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, 2015

[11] C. Southall, C. Wu, A. Lerch, J. Hockman, “*MDB Drums - An Annotated Subset of MedleyDB for Automatic Drum Transcription*”, Proc. of the 18th International Society for Music Information Retrieval Conference (ISMIR), 2017.

[12] L. Breiman, and A. Cutler, “Random Forests”, http://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm