Tutorial for Sound Transcription and Piano Score Following

Stephen Moskal
Rochester Institute of Technology, Rochester, New York 14623

***Abstract*—Lights, videos, and pyrotechnics are all integral parts of a concert that creates an experience like no other. Systems that control these live are primitive, triggering the events based on time or the pitch of an instrument. Detection and transcription of notes from instruments by examining frequencies unassisted can be extremely inaccurate even on single notes due to noise in the recordings and the similarities between notes. This becomes even more troubling when complex chords are played or the tempo is increased to a fast pace. However if a system is trained to the notes beforehand with either single notes or a score with the use of Support Vector Machines, the accuracy of transcription is improved significantly when compared to using just the frequency of the sound as the feature set. This work proposes to train a system using various features of a recording of a score and be able to identify where in the score is playing at any given time live. By identifying the section where the player is playing in a score of music, different events on a stage can be triggered on screen like lights, colors, or videos.**

I. Introduction

For live performances of music, the actual music is the main draw for audiences but also the ambiance, lights, and spectacle of the show plays a major role in the appeal of a live performance. The days of the underground basement rock band performances are still around however there is an increasing demand for performers to have their shows stand out in someway. Whether it is The Flaming Lips with hundreds of balloons, confetti, and gloves with high powered lasers or U2 with massive transforming stages hundreds of feet high, it all adds to the experience. The goal of this work is to create a framework in which performers can input their music into a system and be able to control certain aspects of their show depending on what music they are playing. Most live shows will use some sort of sound based system to trigger lights, pyrotechnics, or videos. However these systems are based on either the elapsed time from the beginning of the song the performance is in, or just a large change in volume. This work instead trains a system before hand to the music and classifies sections of the music meaning that different events can be triggered when a specific note is played.

There are benefits to a system like this as it allows for plenty of flexibility of how a performance can be structured. Using a note based classification system, the performer does not need a technician to start and stop the system in between songs since the time in between songs varies. The performer could be communicating with the audience, taking a drink, or changing their instrument, meanwhile the system is listening for the note that starts the next song. Once that specific note is played, the show automatically begins with no interaction after the system is trained to the set. The scope of this work focuses on piano pieces where the training set is a recording of the song and the system will listen live looking for sections of the recording. However although this paper specifically uses the piano, it can be trained to any instrument including voices or other sounds.

II. System Requirements

One of the key benefits of this design is that it can be used with a wide range of hardwares, ranging from a computer with a microphone or a Raspberry Pi. The programming language Python 2.7 was chosen due to its widespread use, portability, and a massive open source library support. As for hardware, the only requirements is a microphone connected to the machine running the software. The test machine used in this work is a 2013 Macbook Pro using the internal microphone, however other devices like a Raspberry Pi with an external microphone can be used if a more specialized device is desired. All libraries used in this work are open source and freely available.

To begin, three libraries need to be installed for audio streaming, feature extraction, and classification. First is PyAudio[1] which handles all of the interfacing between the program and the microphone. PyAudio can be configured to use any microphone on the system along with reading in sound files from outside of the program. Another benefit of PyAudio is that it allows for multi-threading which will be crucial later on for live streaming of data. Next is the package Aubio[2] which handles taking the data from the file and applying Fast Fourier Transforms and then feature extraction off of the transform. Aubio provides API's that can automatically apply filters to the incoming sounds and organize data for classification. Then lastly, the package Scikit-Learn[3] is used to handle all of the machine learning and classification of the data. Sklearn is an extremely powerful package that allows for supervised training techniques like Classification Trees, Support Vector Machines , etc. along with unsupervised learning techniques like Hidden Markov Modeling and clustering.

III. Implementation

The input of the program is a waveform file (wav) which will act as the training set for the program. The program starts with reading in the sound file and then from there extracting features from the file. Then the data is cleaned up as there are many unwanted features. These features are then assigned a class depending on the frequency of the tone which is used for the classification of the times. Using SKLearn, the test data is then classified using a Support Vector Machine. Once classified, the program then listens in through the microphone in a multi-threaded structure.

[1] http://people.csail.mit.edu/hubert/pyaudio/
[2] http://aubio.org
[3] http://scikit-learn.org/stable/

A. Program Constants

To optimize the running of the program and the classification of the notes there are some tune-able parameters for various functions. For optimizing the program for pitch detection, the window in which the FFT is performed on if can be adjusted if desired. Then when the pitch is calculated the confidence of the value is reported and a parameter can be set to throw out some values if desired. The current tolerance by default is to only accept values with a confidence of .9 or higher. A higher or lower value should be changed depending on the aggressiveness of the data fit desired. When the data is assigned to the classes, there is a small tolerance in Hz that will define a window in which classes are similar. If there are the same note played in the training set multiple times, it will assign those to the same class instead of separate. Lastly, to finely tune the program to a specific range in the spectrum (in this case a piano) constants can be tuned to restrict the classification to just the possible spectrum of the instrument. In the case of a standard piano, it is set to 30Hz for 5000Hz.

B. Feature Selection

Picking the feature set is the most important aspect to get correct for accurate classification. Using Aubio, feature extraction is quite simple. For each window, the FFT of the signal is taken and different functions from Aubio are used to extract features automatically. The first feature to extract is what is called the onset, which is the start a note. This is done by taking the sample and calling the onset method on the Audio onset object. As seen below in Fig. 1, the onset is shown as a vertical line where the raw waveform is the top graph and the amplitude is on the bottom.

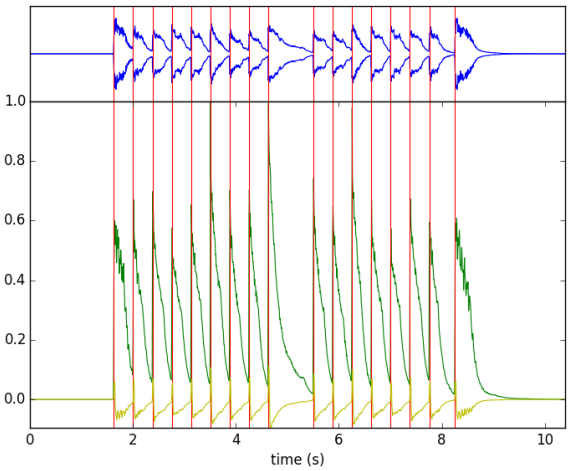

Fig. 1: A graph showing the onsets of each note of a song.

Next for assigning the classes to the data, the pitches of each of the sample are calculated using Aubio again. Using the pitch function with in Aubio, both the estimated pitch and also the confidence of the correctness of the pitch. With the input of the program being a simple scale, Fig. 2 below shows the pitch in Hertz and it's corresponding confidence level for the sample. Classes can be assigned by looking at the frequency in between each onset and assigning a class for similar frequencies.

Because the frequencies of the samples varies so much even in the same section of the note, stronger features that still

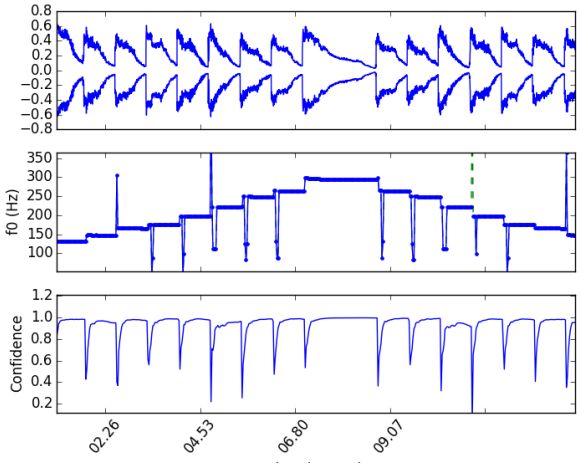

Fig. 2: A graph showing pitches of the sound file over time.

describe the sounds are desired. To do this, the energy at 40 separate bands of the frequency bands are taken over time. This will allow for very accurate classification since even though frequencies of the notes and chords may be very close together, the different parts of the spectrum that note may excite could be very different. This allows for notes to be very separable from each other instead of simply just using the frequency of the transcribed notes. Fig. 3 below shows same data set as before but also showing the energy for 40 different bands of the spectrum.

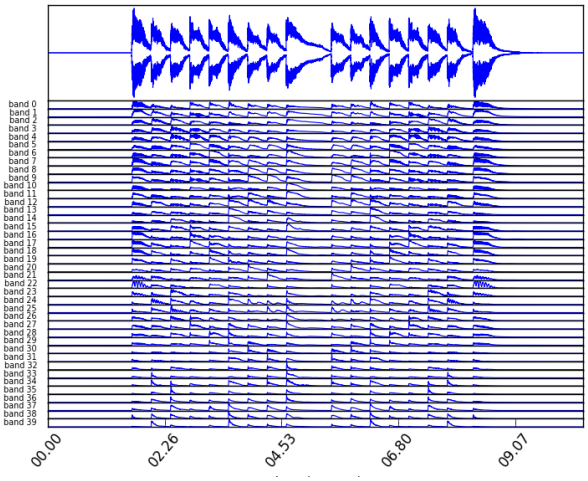

Fig. 3: A graph showing the sound energy for 40 bands in the audio spectrum.

Notice that even though the notes on the piano are close together, the overall energy across the bands are very different. However for the same note the energies are similar to each other meaning that this is a good feature to describe the data. In this work this is the features that will be used for classification but they will need to be organized properly to be used in the classifier.

C. Data Organization

The classifiers in SKLearn require the data to be organized in a fashion where the data used for the X dimensions are

separated from the data in the Y dimension. The X data array describes the features that were extracted from before. In this case the features will be the frequencies and the energies of the sounds. However the data Y, which is the class that the corresponding X data belongs to, needs to be set in a supervised fashion. To do this, the onsets of each of the notes are used. It can be assumed that the samples in between each onset are going to belong to the same class since they are all one note. However there may be similar notes in the set that may be played and those should be assigned to the same class which will reinforce the training. To accomplish this, the average frequency between each onset is found, if some of the samples are below the confidence threshold or outside of the bounds of the set frequency band. Each of these average values between onsets are assigned a unique class and then assigned to the individual samples. If one of the average values are close to another average value (with in the set window) they are set to the same class. Once all the samples are assigned to a class, the *Ydata* array is created. Now both of the arrays are created and then can be inserted into the classifier.

D. Classification

The SKLearn package makes the feat of classification extremely trivial once the type of classification chosen and the data is organized correctly. This work uses the classification technique of Support Vector Machine due to a relatively low amount of data per class but a high amount of features. The robustness of the SVM as compared to a more primitive classification technique like Classification Trees benefits the accuracy and the speed of the system while not providing anymore complexity from a coding perspective. To use the SVM, a simple initialization call of to the SVM import object, svm.SVC(), which returns a classifier object. Then the data is sent into this classifier object to create the model using the method clf.fit(*Xdata, Ydata*). Then to make a prediction the predict function in the same classifier object, clf.predict(*Xtest*), is used with the testing X value. This will return the class in which it predicts that the input belongs to, making it extremely easy to classify and make predictions.

E. Audio Streaming

The last section of the code is the part that handles listening from the microphone live and then making predictions of the note from the stream. PyAudio is used to interface with the microphone and then read in chunks of data. A multi-threaded approach is needed here data needs to be processed on the incoming audio without interrupting the reading in of the audio stream. Thus, there are two threads running simultaneously in this section of the code. One is constantly reading audio from the microphone and appending this data into the back of an array. While another thread is popping data off the top of this array to process and make predictions on the data. This allows for a uninterrupted data stream and processing happening at the same time.

On the side that is popping the data off of the array after it has already been read in from the microphone, the feature extraction is done in the same way except slightly out of order. First, the data is only popped if the array has some thing in it, meaning that it will not process data if it gets ahead of the stream. Then feature extraction only begins when there is an onset detected. If there is an onset it will extract the features exactly before and organize them into an X test array. Then this *Xtest* array is put into the classifier using the predict method and the class is reported back to the user.

IV. Results

The training set used in this work was a single octave (C2-B2) of a grand piano generated from GarageBand in OSX. To test the classifier, the training set included the scale going up the keys and then back down to test to see if the repeated keys are put in the same class. For testing as mentioned before, a MacBook Pro with its internal microphone was used to run the program and an IPhone 5s with the application GarageBand was used to play the piano keys. To start, the keys were pressed going up the scale sequentially starting at C2 and ending at B2. The output of the program for the scale test can be seen below in Fig. 4.

```
Listening for tunes...
Sensed a note:
     Predicted Note:  C2  Freqency:  132.42
Sensed a note:
     Predicted Note:  D2  Freqency:  148.681
Sensed a note:
     Predicted Note:  E2  Freqency:  166.591
Sensed a note:
     Predicted Note:  F2  Freqency:  175.397
Sensed a note:
     Predicted Note:  G2  Freqency:  196.978
Sensed a note:
     Predicted Note:  A2  Freqency:  221.023
Sensed a note:
     Predicted Note:  B2  Freqency:  247.794
```

Fig. 4: Detection results going up the 2nd octave on a piano.

As seen, the predicted note is correct even when dragging a finger across the keys. Note that each of these keys are labeled for demonstration purposes. In this case each of the presses the frequency was correctly extracted however this will be exploited in the next test. To test the invariance of key strike, a single key was pressed 20 times in various different pressures. This is to stress that the current algorithm still correctly classified the keys even when the frequency is miscalculated due to resonance. To ensure that each key was independent the key was muted after classification. The results of this test can be seen below in Fig. 5.

```
Sensed a note:
     Predicted Note:  A2  Freqency:  221.132
Sensed a note:
     Predicted Note:  A2  Freqency:  220.989
Sensed a note:
     Predicted Note:  A2  Freqency:  110.667
Sensed a note:
     Predicted Note:  A2  Freqency:  220.938
Sensed a note:
     Predicted Note:  A2  Freqency:  220.936
```

Fig. 5: Output showing the inconsistency in frequency detection.

If this test was classifying strictly on frequency alone, the 3rd key press in this test would be classified as a much lower

note. However because this work is examining the energies in the frequency band, the key press was classified correctly. The last test in this work tests what happens when a key is pressed that was not a part of the training set. In this case, two known keys were pressed, an unknown key was pressed was pressed twice, and then two more known keys were pressed. The unknown key was F#, which is in between the two known keys pressed. This is to stress that again it is not frequency based, thus it will not get mapped into anything else. The output of this test can be seen below in Fig. 6.

```
Listening for tunes...
Sensed a note:
     Predicted Note:  E2  Freqency:  165.82
Sensed a note:
     Predicted Note:  F2  Freqency:  176.797
Sensed a note:
     Predicted Note:  None  Freqency:  185.976
Sensed a note:
     Predicted Note:  None  Freqency:  185.984
Sensed a note:
     Predicted Note:  G2  Freqency:  196.953
Sensed a note:
     Predicted Note:  A2  Freqency:  221.156
```

Fig. 6: Output showing the effect of pressing an untrained note.

Notice that the F# key was classified to none but the frequency was in between the others. This is again because the energy spectrum of the F# key is very different than that of F2 or G2. The SVM will output class -1 for anything that could not be classified with a significant confidence. That being said, this system does have some issues. Currently the program is over sensitive to outside noise and voices. The program will try to classify the voices and sometimes they do match to some classes. Also the sampling rate may be slightly too slow since the system waits for the start of a note to being classifying. In the case that two notes are played rapidly, the system may skip that note due to the notes not differing in energies enough.

V. Future Work

Two major improvements are in development that are not included in this work. First, a system to listen to a song and trigger events on stage is currently in development. This will be set up as a state machine were the user selects what event that is triggered depending on what sound is played. This creates a time-invariant system that only triggers lights, lasers, or videos when a specific note is played. This is the actual score following portion of the code that is trained to the song or score. This can be adapted to use any instrument for any song.

Second is a more artistic approach to the same problem. Say a performer is using this system to control the lights like mentioned before, however they want to do an unscheduled solo piece. The goal for this system is to have the system controlled by a stomp box on the stage which would triggered solo mode. Solo mode would allow the lights to be directly controlled with no state machine. Say the performer sustains a note and then bends the string aggressively, the lights would change according to the bend of the string. Or if the performer repeatedly bends and unbend the string, the lights will brighten and dim accordingly. This allows for a custom set piece that can be different depending on how the performer plays that night, allowing the performer to tell a story or portray their emotions of that night in a musical and visual manner.

VI. Conclusion

This work shows that using more descriptive features other than frequency allows for a much more accurate classification and transcription of music. It also provided a framework for live transcription and score following for an instrument with the aid of Support Vector Machines. The system needs to be fine tuned more to improve training and optimized to quickly classify notes as it is slightly delayed. However this work provides a good base for score following and was a great proof of concept.