Determining Musical Tonality with Hidden Markov Models in Melodies

Kilian R. McCroskey

April 2022

1 Introduction

Musical tonality can be generally understood as how 'correct', or in tune/key a musical piece is. As such, tonality can be a measure of how pleasantly a piece of music falls on the ear. This project will aim to train a model to be able to classify specific clips of music (MIDI or .mid files) into tonal or atonal categories. This process will involve training Hidden Markov Models (HMMs). The learned detection of musical tonality would be useful, because tonal pieces are sometimes considered to be more pleasing to the ear than their atonal counterparts. Machine tonal classification would assist artists and sound technicians with judging the objective 'correctness' of a piece of music. Implementations could also include training exercises for novice musicians, and possibly provide insights to instrument tuning. E. Gutierrez says, about musical tonality classification: "Manual analysis, intended to obtain ground truth descriptors, becomes very hard when dealing with a high amount of musical pieces, and when the score is not available, as it is the case in most popular music"[2]. Automation of tonality description would solve this issue.

2 Problem

It is currently unclear whether or not there exists a model that can detect tonality in a given, solo-track melody in a MIDI sequence. The goal of this project will be to discover such a pattern that indicates a given audio's musical tonality (if any). I will be training HMM's to render a score based on a piece's tonality, given the single-track MIDI melody. This approach would provide a way to detect musical tonality in an accessible fashion to anyone proficient with musical data.

3 Literature Survey

Work to detect musical tonality in audio files has been done in the past. In 2009, Özgür İzmirli created a model with $> 90\%$ accuracy by using diffusion maps[1]. These diffusion maps were useful in reducing the dimensionality of the dataset Dr. İzmirli had. In the article, the results indicated "The chroma based classification resulted in an average accuracy of 91.2% and the spectrum based classification resulted in 90.4% accuracy" [1]. These different models used pitch data, as well as visual data to classify the audio. This came with downsides, as the spectrum and chroma-based classifiers used geometric models of pitch and key, which is not always obtainable from audio, and can be expensive to compute, and are obtained from "monophonic instrument sounds playing major scales."[1].

Referenced in İzmirli's work is a dissertation by E. Gutierrez. Her thesis, *Tonal Description of Music Audio Signals* reportedly "substantially contributes to the field of computational tonal description"[2, pg.5]. Gutierrez's thesis goes on to say that "There have been many efforts in the literature to model the human cognition of tonality, mainly in the fields of cognitive science and music psychology"[2, pg.20]. This reiterates that we don't fully understand how we perceive tonality in music, and training models to detect tonality could provide insights into how we understand tonal music.

These articles have demonstrated that there are viable ways of detecting musical tonality with and without Machine Learning applications. These have included very calculated approaches to determining pitch, with training sets (e.g. İzmirli's dataset) which do not appear to have audio stored in a typical file format/extension. Instead, in İzmirli's paper, his test set indicates that "Assume the data set containing k elements is given by $X = x0, x1, x2, ..., xk - 1$ with x_i element of R^m."[1, pg.688]. Further, he implements a diagonal normalization matrix in his dataset [1, pg.688]. This appears to be a method of storing the data such that it can be used with diffusion maps. This data would also not be encoded the same way that, say, a .mid (MIDI) file might be encoded. This would make such applications less accessible in assessing tonality by-and-large, and İzmirli acknowledges this. He states, " Since these methods do not scale well with increasing data size, it becomes infeasible to use these methods in online applications"[1, 687].

Discovering research on specifically examining the relationship between MIDI sequences, and binary classification of said audio has proven difficult. Many of the peer-reviewed sources that deal with this use much different methods, and undisclosed data encoding (if any) with specific pitch-detection techniques. For example, Sija Hu covers in her abstract that " In our work, we automate the process of finding the number of states: for a training piece consisting of n PCPs, we initially train an HMM with a large number of states (e.g. n/10) using the Baum-Welch algorithm and then perform an estimate of distances between states."[3] This is a much different approach than I am anticipating, and uses specific techniques to classify "pitch-class-profiles" (PCP)[3]. The Baum-Welch algorithm, in this case, is used to find the unknown variables for the HMM she has trained. This appears to be a common adaptation of said algorithm, and the HMM appilcation on this type of problem seems very appropriate. Given the forward-backward algorithm used in HMM's, the model is highly applicable in contextual problems. Dr. Hu's application of an HMM seems the most logical and accessible for this problem (in terms of MIDI sequences). HMM's will be the most useful for this application.

4 Technical Material

The technical material I had to cover to fully understand this project was no bedtime reading. Since I ended up deciding to implement this with HMM's, I had to rethink and relearn a lot of what I knew about Machine Learning. The strangest thing to grasp is the lack of target properties (Y-value) that would normally be present in a lot of Machine Learning models. Instead, HMM's use an observable 'Markov Process' that is state-dependent. This state-dependency is governed by some hidden state, which, in this case, would be the constructs of musical tonality (which can be incredibly complex and dependent).

In addition to reading up on HMM's, I also had to cover all of the documentation for implementing a training package for HMM's (called HMMLearn, for Python). With this I had to revisit a great deal of material from Dr. Sekoni's CPSC 350 class - which laid my background knowledge for this project. Since this state-dependent model vastly differs from others, I had to create my own forms of validation based on the data I have collected. I particularly enjoyed this, because it gave me a deeper understanding of how these HMM's work, and why they are so useful in this case.

Beyond that, I had to dive a little deeper into regular expressions, string parsing, array slicing, and reshaping with Python. This wasn't terribly difficult, and proved to reaffirm some of my cross-language knowledge of syntax. Specifically, the regular expression aspect of this project bolstered my understanding of how they can be. Writing in PHP for the past few months made the Python of this implementation not so bad. I finished up my learning and review with parameter tuning techniques within machine learning, which was also bizarre in this implementation due to the nature of HMM's.

5 Methodology

In order to source the amount of data I needed to get any kind of meaningful results, I ended up generating .mid (Midifile) melodies from this site. Initially, I expected that this was something I could have generated programmatically; however the source code for this generator was not included with the site. Instead, I manually created 500 samples of these melodies - 250 tonal, and 250 atonal. From there, I segmented the data into 80/10/10 - 80% for training (400), and 10% for both testing and validation (100).

Interestingly enough, I realized a fundamental flaw once I began implementing this solution. Since HMM's are state-dependent, I could not simply use all 400 pieces of training data on one HMM (like a typical ML model). This is because half of the pieces are tonal, and the other half atonal, by design. This would not make sense, because the data given to any one particular HMM is expected to be state-dependent data with a *constant* unobservable, 'hidden' state. This would ultimately confuse the whole process, and likely not yield any consistent scoring. Further, another reason that I could not use one HMM is due to how HMM's are scored. In this particular library, instead of providing a conventional probability $[0, 1]$, a logarithmic probability is returned $(-\infty\ , 0]$. Although this probability can be converted to 'standard' probability simply by using $np.exp()$ (a method that uses Euler's number to calculate the exponent to revert the probability to $[0, 1]$ scale), log probability is a more appropriate way to measure which probability is greater - due to the very fine differences in probability I have observed.

I have decided to go with training two HMM's - for tonality and atonality, respectively. With this logic, both processes are independent Markov sequences, and as such, will be trained on their respective data. My methodology for scoring tonality vs. atonality will be simply comparing the scores of the respective models on the same data point. This can potentially be incorporated into a new, aggregate model.

For this project, I used *Jupyter Notebook* (a python ML development tool). I also implemented the modules 'hmm' (from hmmlearn), and 'MidiFile' (from mido) - both libraries available to the Jupyter IDE.

6 Results

Initially, I tried using a Gaussian HMM (which provides Gaussian emission probabilities that are normally distributed). This was not very successful, and I am not certain what exactly caused the low success of these methods. I am guessing that the normal distribution of the probabilities caused some of the comparative scores of each HMM to be very similar to each other.

I also tried *RandomSearchCV()*, a SciKit Learn function that assists with model selection, providing automation of randomized parameters in an attempt to seek-out a high-scoring model. Unfortunately, this implementation was wholly unsuccessful, as the HMM's lacked a Y-Train value needed for cross-validating, and scoring the models. There was a way to implement a custom scorer, but I could not get this to work in a timely manner. Instead, I experimented with other types of HMM's (Multinomial, specifically), which rendered much better results.

With that, I have successfully produced two HMM's for both tonality and atonality that, when scored against each other, classified tonality correctly 91% of the time. Each HMM was trained in the same way; I was not ambitious enough to try and train them in different ways.

Each HMM is a Multinomial Hidden Markov Model with discrete emissions - meaning that the emission probability for each MIDI track can only take on certain, discrete values, instead of highly calculated probabilities. This is logical, and helpful with classifying tonality - and I see why this training created such a high-scoring set of HMM's when scoring against each other.

Seeing that the two Hidden Markov Models can reasonably classify tonality, it would seem that there is a clear, Hidden Markov process that represents tonality in music, when examining pitch notes in a MIDI file.

7 Future Directions / Limitations

A possible direction this project could go is the route of generating new music files based on the input melodies. This is a feature of the current model I am using, but it is not going to be incorporated into this project. Additionally, more work can be done in expounding the data to other melodies from real-life situations, such as live recording, and real-time interpretation of that data. Either way, this model would enjoy having some more data. Another aforementioned limitation is that I trained both classification HMM's (tonal and atonal) with identical hyperparameters. Experimentation with differing hyperparameters (since tonality and atonality are *technically* two separate Hidden Markov processes) could prove more successful. Generation of new data from the model, and retraining on that model could also prove useful; I have not incorporated that into this project. This project has a humble start, but can potentially turn into a comprehensive way to examine how we interpret music.

8 Sources

[1] Ö. İzmirli. 2009. Tonal-Atonal Classification of Music Audio using Diffusion Maps. 10th International Society for Music Information Retrieval Conference. https://ismir2009.ismir.net/proceedings/PS4-19.pdf. [PDF]

[2] E. Gutierrez. 2006. *Tonal Description of Music Audio Signals.* DEPARTMENT OF TECHNOLOGY OF THE UNIVERSITAT POMPEU FABRA. https://www.tdx.cat/bitstream/handle/10803/7537/tegg.pdf [PDF]

[3] S. Hu. 2006 *Organizing musical pieces by tonal similarity.* Association for Computing Machinery. https://dl.acm.org/doi/10.5555/1127442.1127491

Special thanks to the SciKit Learn machine learning library (in Python implementation).

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. 2011.
Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, Volume 12, pg. 2825-2830.