Machine Learning and Music Composition

Daniel Woeste
Division of Math and Science
University of Minnesota, Morris
Morris, Minnesota, USA 56267
woest015@morris.umn.edu

ABSTRACT

Throughout history, musical composition has been thought of as limited to those gifted with a higher insight. As computers have risen in prominence, they have also asserted themselves as an almost necessary tool used in the field of music. Recent advancements in machine learning technologies has possibly provided a new way for computers to be used in the field of music, as well as possibly bringing music composition to the masses. Machine learning offers the unique possibility of having a program generate the song, leaving the user to edit or pick the song closest to their musical tastes.

Keywords

Random Forests, Markov Chains, Neural Networks, Machine Learning

1. INTRODUCTION

Music has become an almost daily part of our lives, whether it is through background grounds in movies, commercials, or even listening to it for the sake of listening to a song. For many of us, our interactions with music are mostly a passive experience, listening rather than contributing. Recent advancements in machine learning may give people the chance to produce music tailored to their personal music tastes. These programs can be used as musical assistance to enhance or fill in gaps in musical knowledge. They can also be used to autonomously generate music without much input from the end user.

In this paper, I will cover three possible methods that may be used to generate music. These three methods are markov chains, random forests, and neural networks explored by Klinger et al., Ackerman et al., and Johnson, respectively. We will start the paper, in Section 2, by going over the necessary musical and machine learning terminology necessary to understand the larger concepts. From there we move into into Section 3, where I describe exactly what the programs are and the general ideas behind them. After that we progress to Section 4, were we cover how the programs were built and trained to produce music. Finally, we will discuss the actual results of the research in Section 5.

UMM CSci Senior Seminar Conference, November 2017 Morris, MN.

2. BACKGROUND

In this section, we discuss the necessary terminology both musically and that of machine learning to understand concepts throughout the rest of the paper.

2.1 Music

In this section we explain the terminology that is important to the musical side of this paper.

A melodic progression is the interval traveled when changing from one note to another. These intervals are broken down into whole steps and half steps. Combining several progressions together produces a melody. A melody is a combination of pitches, rhythms and note duration, considered pleasing to the ear of the listener. At any point during a piece of music, the most important part played is considered the melody.

Chords are when several notes are being played at the same time. This can have a pleasing effect known as harmony, otherwise known as consonace, or a clashing effect known as dissonance. While most songs tend to be harmonic, dissonance gives the music drive and forward motion. Without dissonance, music lacks a sense of progression. Dissonance, in a simple sense, gives the music a direct path to follow, moving from a clash to peace and harmony

Musical texture also plays an important role in the style of music generated by machine learning programs. Homophonic and polyphonic textures are two types of music textures, which will be discussed in greater detail later. A homophonic texture is a piece of music containing one melody played at a time. The homophonic melody can be played by more than one instrument simultaneously, but there is no competing melody being played at the same moment. Polyphonic means there is more than one melodic lines being played simultaneously. Polyphonic music is a more common style to jazz and contemporary pieces. Simpler classical music and pop music tend to favor homophonic texture due to its easier to understand nature.

2.2 General Framework

All of the methods described in this paper share a similar framework that would be useful to understand. The commonality between all of the methods is that every note is generated in relation to a sequence of previous notes. Sometimes the new note is only dependent on a single previous note, while in other circumstances it depends on a larger set of notes. This method of relying on previous notes allows the program to generate new notes that fit well with the current melodic progression. Similar to how a sentence is formed,

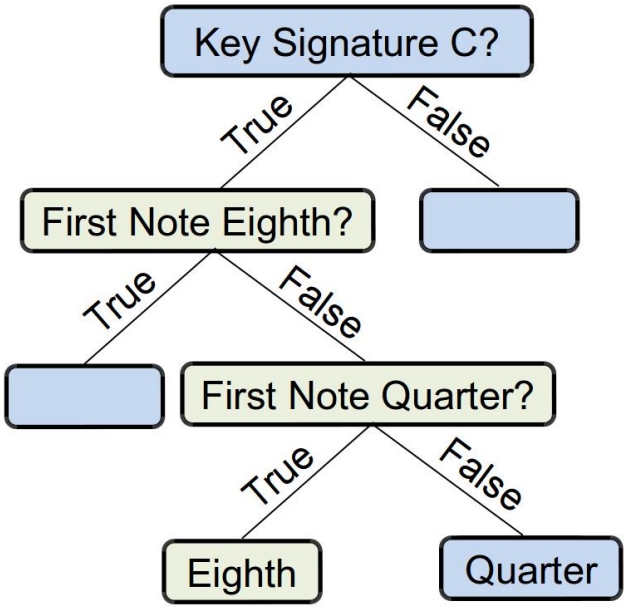

Figure 1: Decision tree example

each new word is dependent on the previous word to make a coherent thought. A coherent melody is created by using notes that blend well with the overall theme already being expressed.

2.3 Machine Learning

Machine learning, a subcategory of artificial intelligence, is an area of computer science that focuses on programs that are able to systematically change their behavior. These learning systems are able to make alterations, or mutations, during a time of training. Alterations to the system are made by using existing musical melodies. These melodies are used as a "final answer" that the program can use to score itself against. Using multiple passes over the training melodies, the program slowly approaches a correct answer by comparing new changes to the answer. After an unknown period of time, the program is able to generate melodies that are similar in style, but not identical to the training data set. We will go over training in more detail in Section 4. Using a machine learning method frees the system from possible biases of the programmer. This style of data driven learning allows machine learning programs to be well suited for applications such as self-driving cars, text prediction, online recommendations, and even the field of music composition.

In the remainder of this section we will describe other basics of machine learning including *decision trees*, *overfitting*, and *finite automata*.

2.3.1 *Decision Trees*

Decision trees are branching structures that represent tests and their possible outcomes, see Figure 1. In this example node, and the corresponding question being asked, are represented by rectangular boxes. The outcomes of these tests/questions are represented by the branches away from the nodes. In this case the outcomes are binary value, another possibility used in Figure 10 is to tie weights to the branches. The decision trees in this paper are read in a top down style, meaning that the tree in Figure 1 would be read

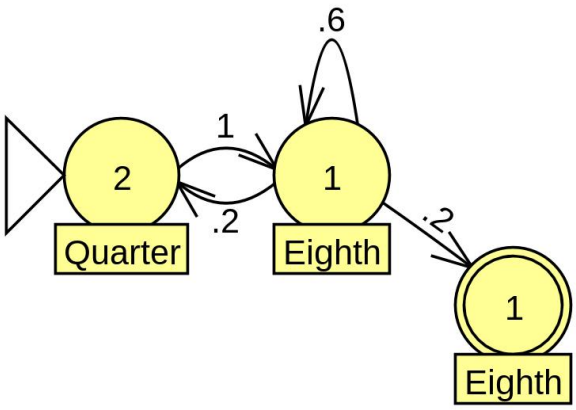

Figure 2: Finite Automata made for an example rhythm in Klinger et al. [2]

by first asking if the piece of music was in the key of C. If answer to that question was a true statement, then we would progress through the tree down the branch indicating true. Following the tree the rest of the way through, we would reach the output of the tree, represented by the eighth and quarter values in the nodes at the lowest level of the tree.

2.3.2 *Overfitting*

A possible side effect with decision trees and all types of machine learning is overfitting. Machine learning algorithms and programs, such as decision trees, develop overfitting because the algorithm itself becomes too complex. Through multiple iterations of training, the algorithm develops too many parameters that are overly specific to the training data set. When this happens the program begins to learn and integrate small fluctuations in the data that are inconsequential to the actual outcome, instead of learning the overall "big picture." A learning system that has developed overfitting cannot be accurately run on data that is not the training data. Overfitting can be prevented by taking a system that has already been trained, and manually altering the program to remove the level of detail that it has. Musically this means that a program developed for composition would only be able to produce the exactly, or almost identical, pieces of music to what it was trained on. It would have failed to grasp the style of music, but instead would only be able to repeat exactly the music it has learned.

2.3.3 *Finite Automata*

Finite automata, otherwise known as finite state machines, are representations of an abstract computational model, see Figure 2. Finite state machines represent a stochastic model of the data. These state machines can only exist in one position at a time. Every possible move is represented by a probability, but the path through the automata is chosen at random, following the probabilities given. In Figure 2, we can see an automata that starts on a quarter note, this node has only one possible move, and that is a 100% chance to move to node labeled eighth. This node now has three possible directions that it could progress, it has 60% chance of returning to itself, and a 20% chance of either progressing back to initial node, or to the final node.

3. METHODS

Now that we have covered the necessary background information, we can now progress onto the three programs introduced earlier in the introduction. These three programs are *ALYSIA*, *markov chains*, and *neural networks.*

3.1 ALYSIA

Automated LYrical Song-writing Application, ALYSIA, is a program that uses random forests to generate melodies for the lyrical inputs. Random forests take the idea of decision trees and expands upon it. A random forest is a large collection of decision tree, and like a real forest each of these decision trees is different.

ALYSIA, developed by Ackerman et al., is a program that is used to generate music for a user provided lyrical input, provided in Music-XML (MXL) form. This is different from the other methods discussed in this paper because it is the only method that deals with lyrics. The other methods only generate arbitrary melodies.

The ALYSIA method relies heavily on function called feature extraction. This gives ALYSIA the ability to look through the MXL files of the user provided lyrics, and the song in the training set, to find musically relevant features. These features can be anything from determining what the key signature of the piece is to looking at the word frequency in the lyrics. There are a total of fifty-nine extractable features [1]. The overall goal of ALYSIA, along with producing music, was to determine what

3.2 Markov Chains

Klinger et al. made a program that uses markov chains to produce music. Markov chains differ from the earlier style, discussed in Section 3.1, in the way that the next note is determined. Markov chains use an entirely a probabilistic model to determine the next note value. Instead of asking a series of questions, the markov chain model used relies solely on the previous note and a set of probabilities to transition to the next note, as seen in Figure 2.

Similar to ALYSIA, Klinger et al. also broke down the generation of music into two categories, rhythm and pitch. The markov process follows a set pattern events. First, a rhythmic pattern is developed. The output of the rhythmic chain is then fed into the markov chain used for assigning pitch value. Due to the nature of pitch values having a name, and an interval the markov chain method broke down the ability to produce pitches into two different methods. These two methods are *absolute* and *relative* markov chains.

The absolute model for the markov chain using direct pitch values from the octave (CDEFGABC). This style of model allows for an easy to read output, because the markov chain is telling you directly what the transitions are. the drawback to this is that it requires more work for the user to have the right format of the input i.e. what key the pitch output should be in.

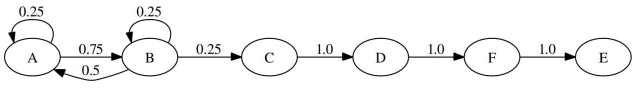

Figure 3: Absolute Model [2]

On the other hand the relative model determines pitch changes by using interval lengths. By using the distance between notes rather than an absolute value of the note, this give the user much more freedom with the output of the relative pitch markov chain. By this we mean that the key, instrument, and position within the octave can be changed at anytime.

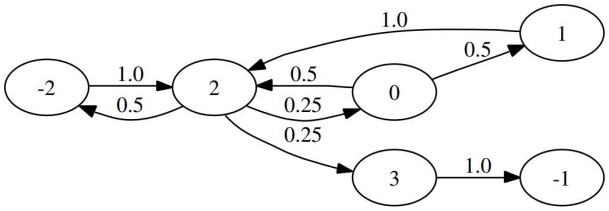

Figure 4: Relative Model [2]

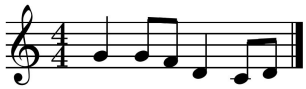

Figure 5: A single bar melody produced after the second iteration of markov chains

The last step to the markov model is to do post processing, known as mutation and recombination. These two ideas take on a couple different forms, *one-point mutation*, *note splitting*, and *recombination*. Even though each of these methods may go about the process differently, they are all working towards the same goal. That goal is to change the melodies made by the markov chain program, after the fact, in order to make it more complex and interesting [2].

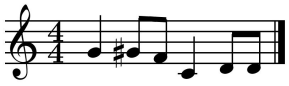

Figure 6: Previous melody after one-point mutation operation

The first of these post processing methods is known as one-point mutation. It is a function that takes the melodic output of the markov chains and traverses over it. As one-point mutation is traversing over the melody, every note that it encounters has a low probability of being raised or lower in pitch value [2]. In Figure 6, we can see that the second note, from the left, has been raised up a half step. This is indicated by the sharp or # symbol.

Note-splitting is a process that is almost identical to the the previous operation, one-point mutation. The difference in the two operations is that instead of changing pitch, note-splitting, changes the duration of the note [2]. In Figure 7, we can see that the second and third notes, both eighth notes, have been split into a run of sixteenth notes. The note values have been cut in half.

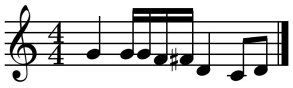

Figure 7: Previous melody after note splitting operation

The final style of post-processing is known as recombination. Recombination is a method of combining two separate melodies, into a new singular idea. It is working towards trying to introduce different melodic ideas into the same piece. For the markov chain program, this would mean that two different melodies would have to be generate. Recombination then takes these melodies and splices them together, without losing any of the original parent melody. The result is a new piece that modulates between the melodic ideas.

3.3 Neural Networks

4. TRAINING AND EVALUATION

Now that we have covered the concepts and ideas behind a couple methods for producing music, we can now move onto how these programs were trained to understand whether or not the music they produced is good or not. Training of these methods relies heavily on being able to evaluate the melodies after each generation.

4.0.1 ALYSIA

4.0.2 Markov Chains

Klinger et al. tried several different methods of evaluation. Two of these that we will talk about in this paper are *Feature Extraction* and *Decision Trees.*

Similar to the feature extraction described for ALYSIA, Klinger et al. is looking for ideas or patterns that could be important for determining the quality of the music. Instead of looking for specific musical parameters, such as key signature, the markov chain method made several functions that calculated ratios out of prominences within the music. Two of these feature extraction are *rests on downbeats,* and *repeated pitch.*

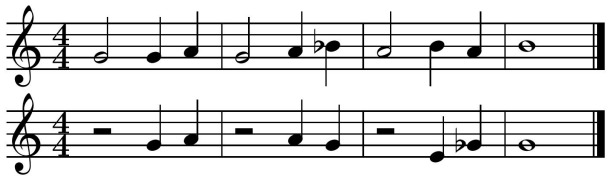

Figure 8: *Rests on Downbeats* with a value of 0 for the first melody and 1 for the second [2]

The feature rests on downbeats determines the ratio between the number of downbeats and the number of downbeats on which there is a rest. In Figure 8 are two melodies with 4 downbeats: One with a value of 0 with no rests and one with 3 rests and a resulting value of 0.75 [2].

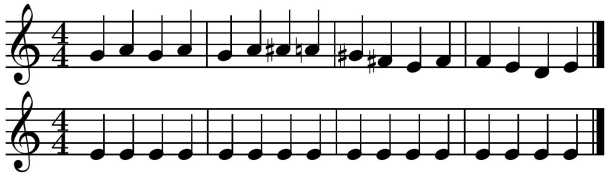

Figure 9: *Repeated Pitch* with a value of 0 in the first melody and 1 in the second [2]

The feature repeated pitch computes the ratio between the number of all intervals with a size of 0 and the number of all intervals (= number of notes - 1). In the example in Figure 9 is one melody without pitch repetition (value 0) and one with all possible pitch repetitions (value 1) [2].

Klinger et al. then tried an evaluation method using decision trees. Using the Weka-library in JAVA, Klinger et al. built a variety of decision trees with different numbers of fitness classes, the final score. The decision tree in Figure 10 we can see a tree that has two fitness classes, 0 and 10. This tree was built as an extension to feature extraction. By providing the decision trees with these features, the decision tree is then able to string the questions together to find series of questions that are important.

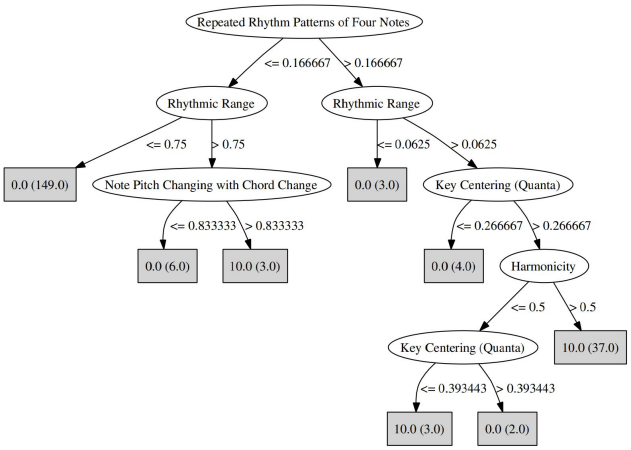

Figure 10: Example of a decision tree used to evaluate melodic outputs for markov chains [2]

If the feature Repeated Rhythm Patterns of Four Notes is very small and Rhythmic Range is also not very high the individual is classified as a bad melody. But if the Rhythmic Range is high and the Note Pitch Changing with Chord Change is also high it is classified as being a good one. Likely the following explanation holds: In the examples the chords are often changing with the bars. So if the rhythmic range is high there is a good possibility that the rhythm is confusing, but if there is always a note on the first beat in a bar it is considered not bad [2].

4.0.3 Neural Networks

5. RESULTS

In Figure 11 we can see a comparison of three decision trees with different numbers of fitness classes and levels of

pruning. Pruning is the process of removing section of the decision tree that are ineffective at classifying fitness. The classification of the one with eleven fitness classes is identical using a pruned and an unpruned variant so there is no difference. The individuals a to l are sorted with respect to the interactive evaluation. The tree with eleven fitness classes makes some errors on the bad individuals and the unpruned one with five fitness classes is questionable in the middle fitness area. The pruned tree with five fitness classes leaves the best impression [**?**].

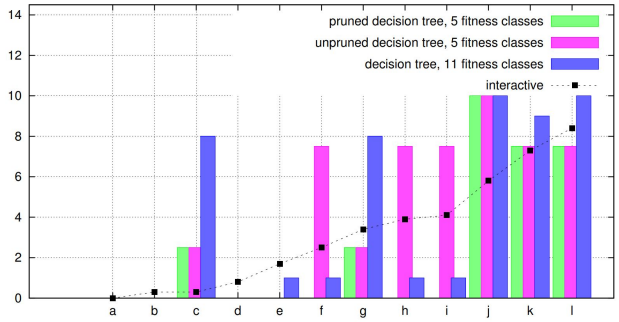

Figure 11: Markov chain results plotting decision tree vs interactive evaluation [2]

6. REFERENCES

[1] L. Ackerman. Algorithmic songwriting with alysia. *EvoMusArt*, pages 1–16, March 2017.

[2] R. Klinger. Automatic composition of music with methods of computational intelligence. *WSEAS TRANS*, pages 1–16, March.