**CS 478 Group Progress Report**

**Nick Angell, Iain Lee, Brian Rasmussen, Anne Thomas**

**The Problem**

The goal of our project is to classify the time period of a piece of classical music based on the notes themselves (rather than metadata). In other words, we want to see if patterns specific to each time period’s style are recognizable by a machine learning model. This would allow for an effective comparison between different pieces of music within each time period.

The most famous periods of classical music fall into the common-practice period, which include the *Baroque* (1600-1750), *Classical* (1750-1820), and *Romantic* (1780-1910) eras. Music which adopts later styles fall into the *20th Century* (1901-2000). Because composers within each period adopted styles pertaining to their respective period, each piece of music should be classifiable from the music itself. In order for the model to generalize these styles, the model will be analyzing the notes themselves, effectively “listening” to the music, rather than analyzing the composer or year composed.

**Our Machine Learning Model**

We have decided to use a Recurrent Neural Network in order to classify each piece. The RNN works by looking at a series of time steps and then outputting it’s prediction of most likely time period at the end. Each time step takes as input a one hot encoding of notes and some weights from the last time step. The RNN works because of the weights that persist from time step to time step, with the output of every time step being affected by the previous time steps. We chose the RNN because of the influence of time as well as the RNN’s inherent ability to deal with inputs with varying length (meaning each input song could have a different length/number of time steps). We seperate each piece of music into a series of time steps and then train the network on each of those pieces.

The other machine learning models that could be useful here might be an MLP backpropagation network and a K-Nearest Neighbors approach. In order for the MLP to work, though, we would somehow need to combine all of the time steps and notes in a piece of music into one feature space. This would be possible but we are limited because then each piece of music would have to have the same number of notes and time steps. You also lose the time information the is inherent in the RNN. The KNN could work by measuring similarities in notes and speed, but again we lose time information and we feel like it would be less accurate. However, we might come back to this if we feel the need to.

**How we get Data**

We originally were planning on pulling from databases/websites that had a wide variety of songs, types in Midi file. The problem that we encountered is that most of these databases require some kind of membership. Being poor college students we can not pay for some membership to use these databases/websites. One website that we found that meets are required data type is the website <http://www.piano-midi.de> this website has songs from the different eras that we want but is limited to only piano. This should not be a problem because we want this model to predict more based on the notes and not on what instruments were used.

The Midi Files will be downloaded from the above website. Because the Midi File format is harder to work with we will be converting from plain text format to JavaScript Object Notation (JSON) . By switching to JSON format we will be able to more easily access the data that we need to use to train the model. We will be using an online tool that was found on github called ToneJs MidiConverter (https://github.com/Tonejs/MidiConvert) . To make this procedure easier. What this softwares does is convert from Text format to JSON format. To make this procedure easier we will be Automating the ToneS Midi Converter to convert all Midi files into the friendly easy to use JSON version of each song. From the JSON version of each song we will easily be able to make the one hot encoding vectors for each song.

We are downloading MIDI files, which are a standard digital representation of musical sound, from

**Instances**

We define an instance as the first 30 seconds of a piece of music. Among other metadata, the MIDI files contain information on the start time, pitch, velocity, and duration of each note. We use start time to determine the timestep of the input going into the recurrent neural network; notes with the same start time will be included in the same timestep. The input of each timestep will include the pitch(es), velocity, and duration of the notes. Pitch is a nominal value representing the note on the scale and its octave; the MIDI files have a range of 119 pitches but we are only interested in a range of 95, which we are representing with a one-hot encoding. Velocity is a real-valued index for the volume as well as the attack, or intensity, of the note, which can tell us (or the network) about the tone and musical phrasing. Duration is a real-valued time in seconds, and can tell us (or the network) about the speed and density of the notes. For example, for the following json encoding of two notes, there will be two timestep inputs given to the neural network:

{

"name": "G4",

"midi": 67,

"time": 0.40709468958333334,

"velocity": 0.4409448818897638,

"duration": 0.20270275

},

{

"name": "C5",

"midi": 72,

"time": 0.6097974395833333,

"velocity": 0.47244094488188976,

"duration": 0.20270275000000004

}

The input will be formatted as follows, with the one hot encoding of the pitch followed by the velocity and duration (we ignore the midi value in the file);

Time step 1: [[

0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. **1**. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.

0.4409448818897638 0.20270275 ]]

Time step 2: [[

0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.

**1**. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.

0.47244094488188976 0.20270275000000004 ]]

We plan to download midi files for around 50 pieces per time period, with the goal of representing several pieces from each of at least 10 composers in each period.

Future Plans

* Already done: basic one-hot encoding and initial testing
* Integrating midi-json converter code into workflow
* Merge and order tracks
* Download and label all instances
* Start iteratively testing model
* Possible future directions: binning similar start times, pieces with additional instruments
* Improving the model and/or trying other models