Music Generation -- Component Spec

Overview

This project is built using the Python package for music generation **Magenta** and MIDI/MusicXML preprocessor **Music21**. The purpose of this project is to assist with the music generation process by preprocessing MIDI or MusicXML files to make current models generate better sounding and original music. This is built for content creators looking for original music for their content, as well as anyone interested in simple music generation tools.

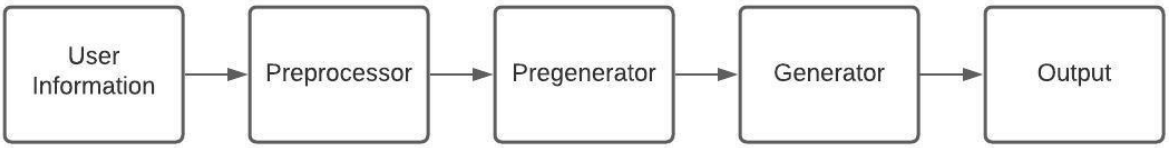
Reference Documents

1. Packages that will be used in this project
    a. Magenta: https://magenta.tensorflow.org/
    b. Music21: http://web.mit.edu/music21/
    c. LAKH MIDI dataset: https://colinraffel.com/projects/lmd/

Software Components

High level description of the software components such as: data manager, which provides a simplified interface to your data and provides application specific features (e.g., querying data

subsets); and visualization manager, which displays data frames as a plot. Describe at least 3 components specifying: what it does, inputs it requires, and outputs it provides.



Preprocessor
The purpose of this module is to perform preprocessing to make decisions about music composition. In fact, it would be possible to create music solely using the preprocessing module, but the decisions from this module will be passed along to the generator in later steps of the pipeline.

As soon as the user passes information to our package, it first reaches the preprocessing module. It expects a directory of files to be analyzed. The directory of files will be iterated over, parsed, and calls to submodules will be made to analyze the files provided. This step is to prevent leakage between modules or submodules that need the file objects produced from parsing. The expected output of this module is 2 stochastic matrices which will be passed to the pregenerator to generate chords and melodies using Markov chains.

The submodules contained in the preprocessor module can be separated into Note Distribution, Chord Distribution, and Preprocessor. The Note Distribution submodule analyzes the music files passed into it and finds the probability between neighboring musical notes across all files to generate a stochastic matrix that will be passed to the pregenerator. In a similar manner, the Chord Distribution submodule also produces a stochastic matrix, but some additional processing is necessary. This submodule converts all notes that occur simultaneously into chords. Let's say, for example, a guitar and violin are playing at the same time. Instead of treating them as two separate instruments, we collapse them down into a single instrument and consider the notes being played at the same time as musical chords. We find the probability between neighboring chords and produce a stochastic matrix that gets passed into the pregenerator. Finally, the Preprocessor submodule parses the music files (MIDI or MusicXML) provided by the user and it calls both submodules previously mentioned (Notes Distribution and Chord Distribution) to operate on the parsed data.

Pregenerator
The Pregenerator module is responsible for deciding the backing chords and primer melody that will be fed into the Generator. At this point, we have established the stochastic matrices of the notes and chords from the example files provided by the user. Now, the Markov submodule will use those stochastic matrices and, through Markov chains, generates both the backing chords and primer melody. The expected output of this module are two strings that will be fed into another pregenerator submodule called the Magenta File Generator. The two strings created

from the Markov submodule will be used to create the corresponding Magenta file necessary for the Generator module to create better sounding music. In particular, this is the point when the backing chords and the primer melody generated from the preprocessor will actually be inserted.

Generator
The generator has 2 submodules: Analyze and Generate. The Analyze submodule takes as input the music sample files provided by the user and instantiates matrices with notes and chord distributions. The Generate submodule takes 2 arguments as input: bars (which determines the length of the output song) and output directory path. Inside the method, the matrices with the notes and chord distributions are used to generate the primer melody and primer chords using our Markov chain from the Pregenerator module. Those primers are then passed to a Magenta pre-trained Polyphony RNN model that will continue composing the song until it completes the designated number of bars for the song. Finally, the Generator will output the composition as a MIDI file into the output directory path specified by the user. The user then will be able to take that file into any Digital Audio Workstation to use additional tools to make the audio sound better.

Component Interactions

The interactions of our project will be based entirely in a Python package and people can generate their own music using our package. We decided that the interface of our package would require 3 arguments that the user will provide. The first argument is used on the analyze module and it is the music samples. The second argument, used in the generate submodule, is the amount of bars the music should be, which allows the user to control the length of the generated music. The third argument expected from the user is a directory of the MIDI or MusicXML files that our preprocessing module will analyze. An example of this interaction is presented below.

```
emg_obj = emg.EasyMusicGenerator()
emg_obj.analyze('music/')
emg. obj.generate(10, ‘output/’)
```

Let’s now observe some use cases of our package.

1) **Content Creator Looking For Music**

Recently, a major cause for concern among content creators is DMCA restrictions on the use of copyrighted material in digital content. However, many content creators still wish to have music included in their content. Our user, however, doesn't want to select from the music library that millions of other content creators choose from. So, the user installs our python package and creates their own music. In this case, we would expect that the user will provide our package with a directory filled with example material that our preprocessor will analyze and generate the corresponding stochastic matrices. These are passed to the pregenerator which generates the strings necessary for the generator to produce better music. The generator then generates the music and from there, the user can listen to the generated music and decide whether it is acceptable for their content. If not, the user can attempt to provide more example material, or, due to the randomness of music generation, attempt to regenerate the music on the same set of files in hopes of better results.

2) **The User is Interested in Music Generation Techniques**

Another expected use case is someone who is generally interested in music generation techniques and would like to have a simple interface with Magenta to work with. The same process described above would occur, but, of course, with different stochastic matrices and outputs by all components of our pipeline, hopefully to the users liking.

Development Plan

Week 1:

- Build and structure github repository
- Data gathering and cleaning from two data sources
- Pre-process training data

Week 2:

- Create technology review of potential python libraries that we could use
- Choose libraries that we will use in our project
- Test the use of chosen libraries
- Choose music parameters we want to work with
- Write functional and component specification documents
- Generate music in any form

Week 3:

- Create setup.py file
- Read in pre-processed data into python module

- Chose main module we will be working with
- Code a model to generate music with Magenta
- Adapt code to include parameters with user specifications
- Start continuous integration with travis-CI or Github Actions
- Create unit tests of specific use cases

Week 4:

- Incorporate feedback into code
- Leave example use case in main function to show how our package can be used
- Create final powerpoint presentation

Testing

The testing required for this package will happen at each component and subcomponent, following the testing guidelines taught in the course.

Preprocessor
The preprocessing module will need to test whether files are of the correct format with regards to both filetype (MIDI and MusicXML) and the contents of the file inside. In particular, a file must contain notes which the preprocessor can analyze. In addition, basic file I/O tests should be made, such as FileNotFoundExceptions. This could be an issue for inexperienced users that aren't familiar with entering file paths into python packages.

Pregenerator
Testing will need to be conducted to see whether the Markov chains are being generated properly. Special attention should be made to see whether the probability of notes/chords are being properly represented in the stochastic matrices used in the Markov chains. Testing should be conducted to ensure that the Magenta file corresponding to the produced backing chords and primer melody are created correctly.

Generator
Testing for the generator could include ensuring that the generated music file is of a correct format such that the user could easily insert the file into a Digital Audio Workstation. Essentially the same testing conducted for the Preprocessor files could be used for testing the generated files.