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.

Data

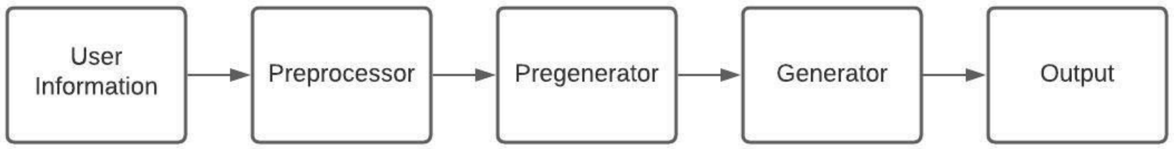
The primary dataset that we will be using for our project comes from the Multitrack Contrapuntal Music Archive which features 475 pieces composed by famous Baroque Classical composers(Vivaldi, Albinoni, J.S. Bach). The data is in MusicXML form which follows a regular XML type of structure, with tag names corresponding to different types of data understood by the MusicXML protocol. An example of MusicXML is shown below.

```xml
<score-partwise version="3.1">
  <part-list>
    <score-part id="P1">
      <part-name>Music</part-name>
    </score-part>
  </part-list>
  <part id="P1">
    <measure number="1">
      <attributes>
        <divisions>1</divisions>
        <key>
          <fifths>0</fifths>
        </key>
        <time>
          <beats>4</beats>
          <beat-type>4</beat-type>
        </time>
        <clef>
          <sign>G</sign>
          <line>2</line>
        </clef>
      </attributes>
      <note>
        <pitch>
          <step>C</step>
          <octave>4</octave>
        </pitch>
        <duration>4</duration>
        <type>whole</type>
      </note>
    </measure>
  </part>
</score-partwise>
```

Reference Documents

1. Link to Functional Specification
2. Packages that will be used in this project
    a. Magenta: https://magenta.tensorflow.org/
    b. Music21: http://web.mit.edu/music21/
    c. Pyplot: https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.html

Software Components



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.

The submodules contained in the preprocessor module can be separated into Note Analysis and Chord Analysis. The Note Analysis submodule analyzes the music files passed into it and finds the distribution of musical notes across all files. In a similar manner, the Chord Analysis submodule converts all notes 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 as musical chords. We find the distribution of these chords across all files.

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 distribution of the notes and chords from the example files provided by the user. Now, the Markov submodule will use Markov chains to generate 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
From the user provided arguments, and combination of everything that has happened in the pipeline so far, the model will then generate a melody over the backing chords based on the primer melody. A MIDI file will be generated in the same working directory as the user instantiated our project. 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, but if time permits, a web-based interface could be developed to easily allow people to generate their own music using our package. We decided that the interface of our package would require 4 arguments that the user will provide. The first is the tempo of the music. Surprisingly, tempo is not a necessary requirement for music generation from Magenta, so we allow the user to decide what tempo the generated music should be. The second argument is the amount of bars the music should be. A combination of the tempo and bars 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. The final argument expected by the user is an output format. The final result for this argument could be represented as a boolean argument such as 'isMidi = True.' This will allow the user to choose between using MIDI or MusicXML as an output format. An example of this interaction is presented below.

```
music_generator.generate( tempo, length, output_file,
output_format)
```

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. 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. Our final project will ship with a dataset which our model will analyze and generate on, so it isn't necessary that the user will have to provide their own data set. So, this will be an easy interface for users interested in music generation without the hassle of setting up magenta files for music generation as one would normally have to.

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
- Build user-friendly platform (either deep module with hidden implementation, or a front end interface)

Week 4:

- Wrap up user-friendly platform
- Test platform with potential users and get feedback from user acceptance testing
- 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 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.