COMP3717 Introduction to Music Processing

Coursework Final Report

Student: cxwn84

Table of Contents

1. Introduction

1.1. Goal

This system's goal is to generate novel musical output as MIDI files which can be played on VLC. The aim is to measure how different music generation techniques compare to real music.

1.2. Techniques

Since pop music follows a relatively simple structure, I chose the n-gram sequential model. Given a sequence of the previous n-1 events, we can calculate the probability of the current event. By counting all the bigrams (n=2), I can create a transition table that tells me the likelihood of the next event.

After generating music, I used pitch-class profiling for qualitative evaluation. A pitch class can be described as a set of octave and enharmonically equivalent notes [1]. Thus, every score can be analysed to produce the distribution of notes across the twelve pitch classes in integer notation (0 to 11).

1.3. Dataset

This project uses the CoCoPops dataset [2] which contains melodic transcriptions in humdrum format. Testing revealed that all the notes in the RollingStone sub-corpus had a quarterLength of 1, which would have imparted a monotonous rhythm on my model. Thus, my models were only trained on the 214 songs in the Billboard sub-corpus.

2. Approach

2.1. Basic: Count Probability Model

As a starting point, the basic model follows the approach from Mark's practicals. In Figure 1, the occurrence of MIDI numbers in every song is summed into a single Counter dictionary. This is repeated for the note durations (quarterLengths). These Counters are used as options and weights to generate random sequences of note pitches and durations, which are then saved as MIDI files.

My main modification is the *make_random_sequence_qls()* function which ensures that the random sequence of quarterLengths sum up to my desired song duration (e.g. 1 minute).

```
Counter({62: 9012, 64: 7757, 59: 6203, 60: 5895, 57: 5655, 66: 4970, 67: 4436, 65: 4370, 61: 4117, 69: 3733, 55: 3641, 63: 3191, 58: 290
9, 54: 2366, 68: 2333, 52: 1948, 70: 1771, 56: 1699, 71: 1615, 72: 1311, 53: 1301, 50: 797, 74: 725, 73: 683, 49: 638, 51: 594, 48: 488,
46: 369, 47: 359, 44: 317, 75: 232, 76: 182, 45: 161, 42: 105, 43: 76, 77: 61, 79: 55, 78: 40, 41: 25, 80: 17, 81: 16, 36: 15, 82: 9, 40:
8, 38: 8, 37: 7, 83: 1, 39: 1})
```

Figure 1: Summed Counter of MIDI pitches

2.2. Intermediate: Single Bigram Model for Pitch

My intermediate model uses the naive and smoothing n-gram implementations from Robert's practicals. In Figure 2, the first step was to determine the total pitch range of the sub-corpus.

```
The smallest MIDI number in this set of songs is: 36
The biggest MIDI number in this set of songs is: 83
Parsed through 214 songs.
```

Figure 2: Finding the smallest and largest MIDI numbers

With this knowledge, I created a 48x48 numpy array to store the transition probabilities of MIDI notes 36 to 83. For each song, I extracted the pitch sequence as MIDI numbers and used the

smoothing n-gram model with n=2 to generate a bigram matrix. I also introduce a prior count of 1 to deal with unknown contexts.

Figure 3 shows how each song's bigram matrix maps to my 48x48 transition table. I add every value in the bigram matrix to its corresponding cell in the transition table and repeat this for all 214 songs. Although the probabilities do not sum to 1, they are still weighted relatively and perform as intended on the Python random.choices() function.

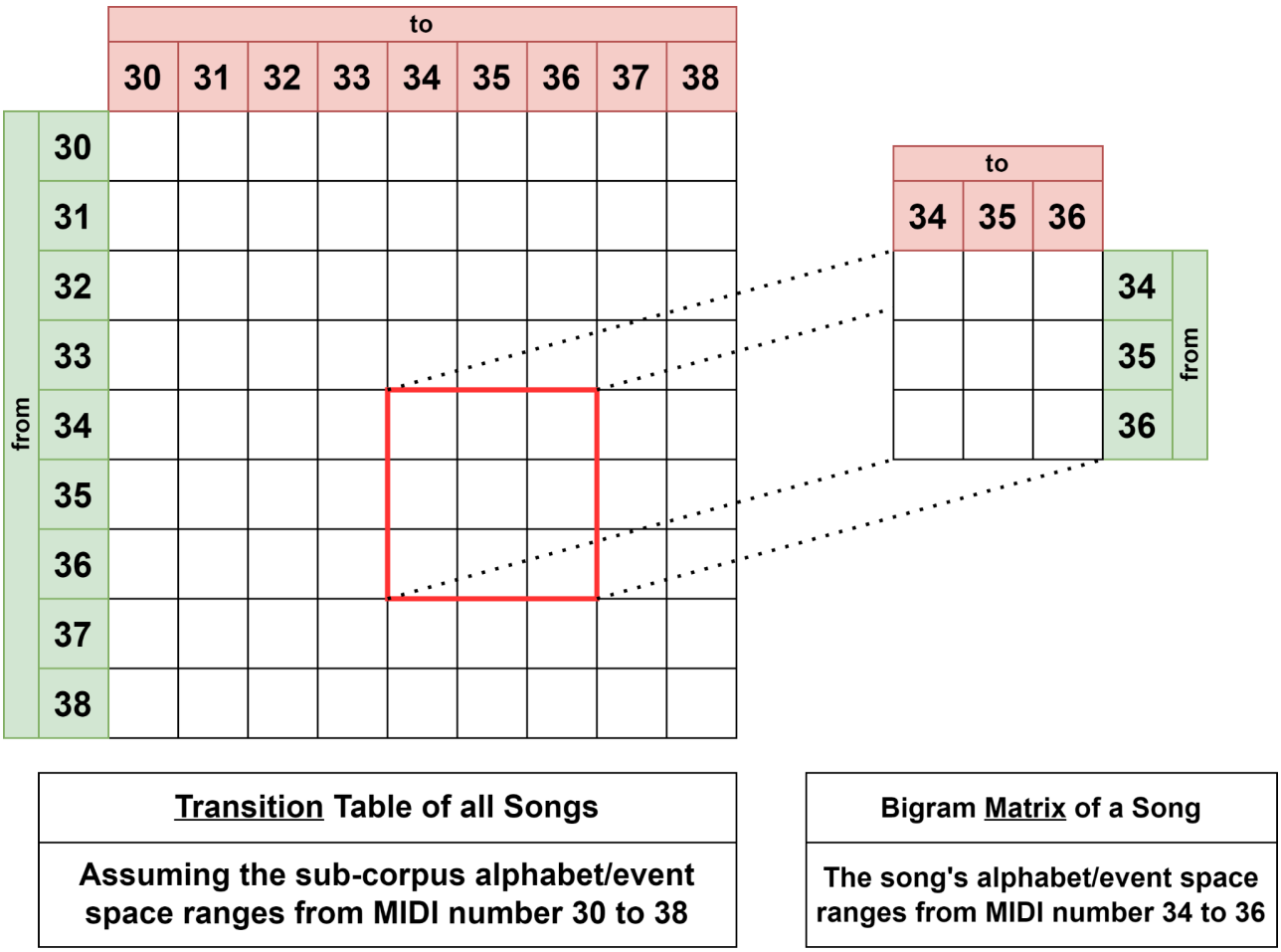

Figure 3: How a single song's bigram matrix maps to the transition table

For generation, *sequential_random_midis()* uses a starting MIDI note and the weights in the transition table to create a pitch sequence that exhibits the patterns captured by the bigram model.

2.3. Advanced: Double Bigram Model for Pitch & Duration

The intermediate approach still generated note durations using count probabilities. Thus, my advanced approach uses duration bigrams to capture the sequential patterns of quarterLengths. In Figure 4, we can see that there are many rarely occurring note durations.

```
Counter({0.5: 46491, 1.0: 15615, 0.25: 12807, 1.5: 2918, 2.0: 1944, 0.75: 1888, 0.0: 1798, 4.0: 603, Fraction(2, 3): 568, 3.0: 524, Fraction(1, 3): 514, 0.125: 239, 0.375: 124, Fraction(1, 6): 93, 3.5: 25, 6.0: 25, 1.75: 20, Fraction(4, 3): 14, Fraction(2, 5): 10, 0.0625: 1, 0.1875: 1})
```

Figure 4: Counts of note durations in quarterLength

To reduce the spatial overhead of durations with extremely low probabilities, I employed a binning strategy to convert rare quarterLengths to their nearest common duration as shown in Figure 5.

```
if element.quarterLength == Fraction(2,3):
    qls.append(0.75)
elif element.quarterLength in (Fraction(1,3), Fraction(1,6), 0.375, 0.125, 0.0625, 0.1875):
    qls.append(0.25)
elif element.quarterLength == 1.75:
    qls.append(2)
elif element.quarterLength == 3.5:
    qls.append(3)
elif element.quarterLength == Fraction(4, 3):
    qls.append(1.5)
elif element.quarterLength == Fraction(2, 5):
    qls.append(0.5)
else:
    qls.append(element.quarterLength)
```

Figure 5: Binning quarterLengths when analysing duration sequence

To reuse the n-gram model implementations that were designed to take MIDI integers as the alphabet, as shown in Figure 6, I wrote the *ql_sequence_to_ints()* function to make a duration sequence compatible with existing code.

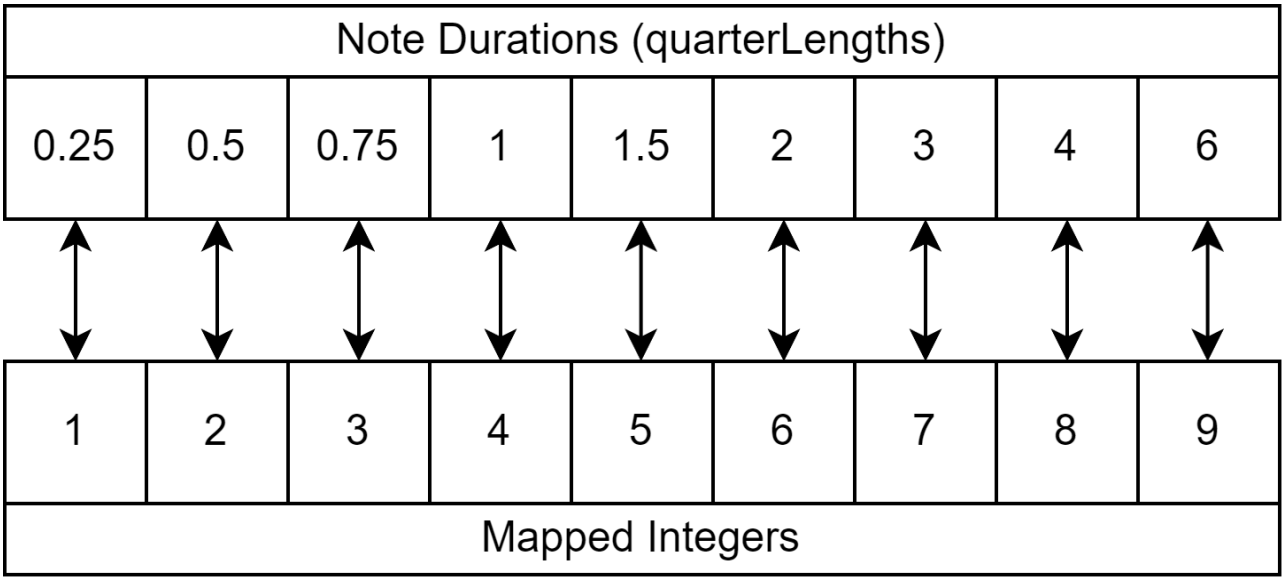

Figure 6: Nine possible note durations are mapped to integers 1 to 9

Lastly, I created another transition table for note durations and summed the values in the bigram matrices using the same mapping process outlined in Figure 3. For music generation, I provided a starting MIDI number and quarterLength. The helper functions *sequential_random_qls()* and *sequential_random_midis()* then use their respective transition tables to generate sequences that are saved as a MIDI file.

3. Evaluation

For evaluation, I applied both qualitative and quantitative methods based on a paper by Wu & Yang [3]. The primary goal is to measure how closely my pieces resemble real music, while the secondary goal is to identify if any correlation exists between human-perceived data and theoretical metrics.

3.1. Listening Test

I prepared pairs of 1-minute pieces for basic, intermediate, advanced, and real songs. A blind listening test was conducted on friends to score eight pieces on the following metrics: Overall Quality (O), Impression (I), Structuredness (S), and Richness (R).

The responses in Figure 7 show that none of my approaches come close to the quality of real music. However, the combination of pitch and duration bigrams in my advanced approach led to scores that were higher than my other approaches in every metric. Thus, my efforts to increase algorithm complexity were slightly successful. In the future, I think incorporating longer n-gram sequences would further increase the Structuredness of my compositions.

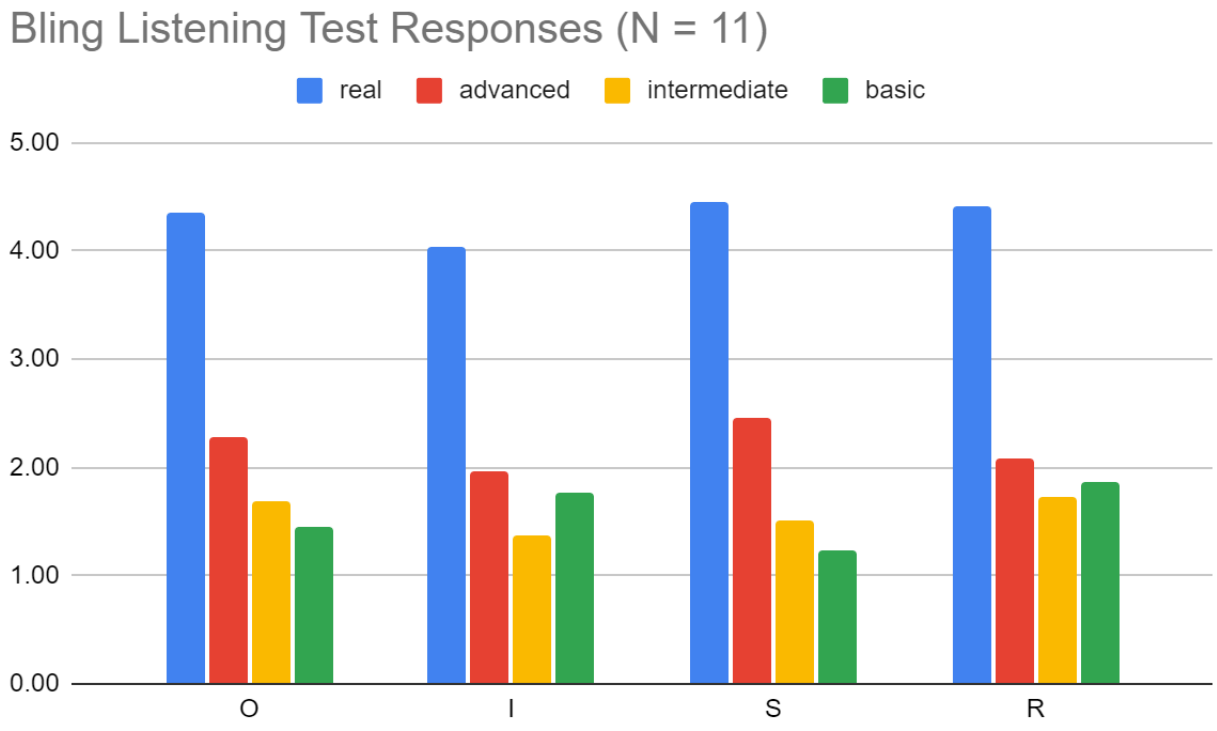

Figure 7: Listening Test Responses

3.2. Pitch Class Profile (PCP) Entropy Analysis

Entropy measures the “uncertainty” of a probability distribution [4]. Wu & Yang suggested that if a piece of music has a clear tonality, then the PCP should be low-entropy [3]. In the following equation, C_i is the normalised count C for pitch class i in integer notation:

$$Entropy(PCP) = -\sum_{i=0}^{11} C_i \log_2(C_i)$$

Figure 8 shows the PCP entropies of the pieces in the listening test as well as the sub-corpus mean. The results show that my approaches have much higher entropies, suggesting that their tonality is inferior in clarity compared to real music.

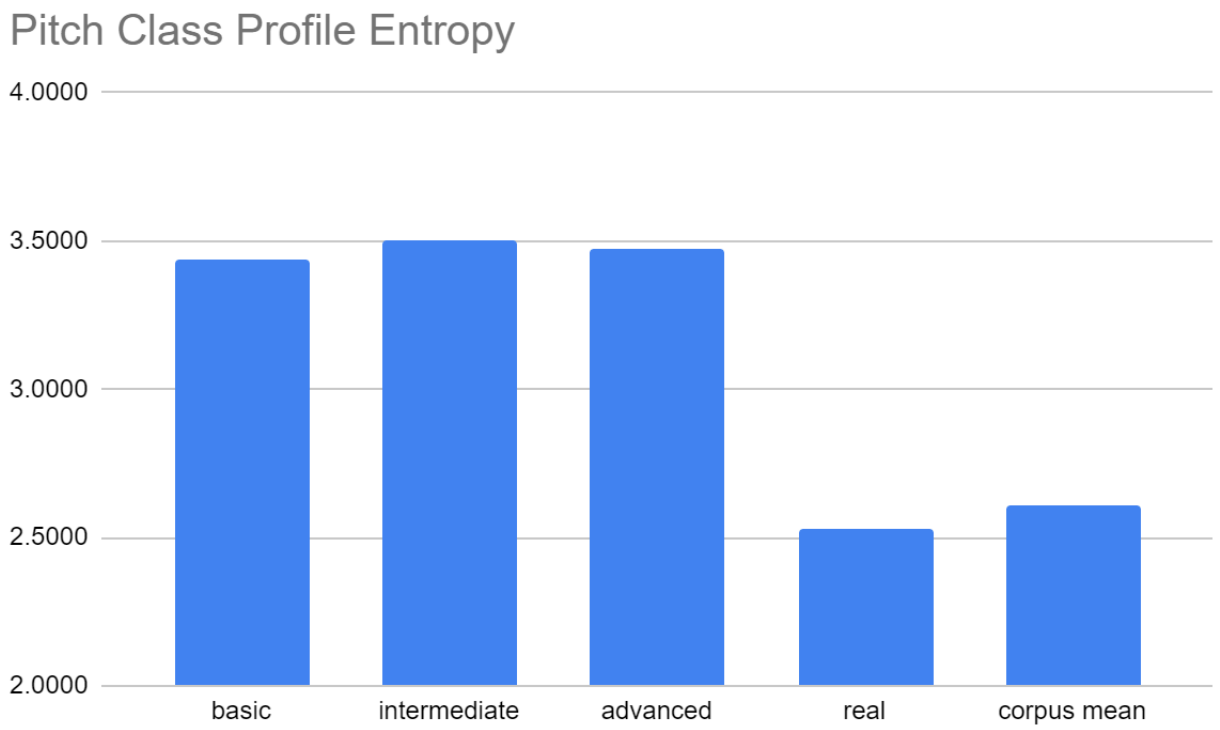

Figure 8: Entropy Comparison

It is interesting that despite my approaches having similar entropies, there is a perceivable variance in quality by my human listeners (Figure 7). In the future, I would try to quantitatively assess rhythmic quality and chord progressions for a more holistic quantitative analysis.

(999 Words)

4. References

[1] M. Lavengood, “Pitch and Pitch Class” in *Open Music Theory*, 2023. Version 2. [Online]. Available: https://viva.pressbooks.pub/openmusictheory/chapter/pitch-and-pitch-class/

[2] C. Arthur and N. Condit-Schultz, “The Coordinated Corpus of Popular Musics (CoCoPops): A meta-corpus of melodic and harmonic transcriptions,” in *Proc. 24th Int. Soc. for Music Information Retrieval Conf. (ISMIR)*, Milan, Italy, 2023. [Online].
Available: https://github.com/Computational-Cognitive-Musicology-Lab/CoCoPops

[3] S.L. Wu and Y.H. Yang, “The Jazz Transformer on the front line: Exploring the shortcomings of AI-composed music through quantitative measures,” 2020. [Online].
Available: https://arxiv.org/pdf/2008.01307.pdf

[4] C.E. Shannon, “A mathematical theory of communication”, in *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379-423, 1948. [Online].
Available: https://people.math.harvard.edu/~ctm/home/text/others/shannon/entropy/entropy.pdf