MAV PROJECT REPORT

Source Code: Google Collab (https://shorturl.at/kkjkS)

Topic: Using Markov Chains to Analyze and Generate Music Compositions

Introduction & References

INTRODUCTION

This project applies Markov chains and linear algebra to analyze and compose classical music. By modeling notes and durations as probabilistic states, we build transition matrices to capture melodic patterns and generate new compositions. Using first-order Markov chains derived from classical music datasets, we calculate transition probabilities to model musical structure. These models predict notes and durations, enabling the creation of melodies that reflect classical styles.

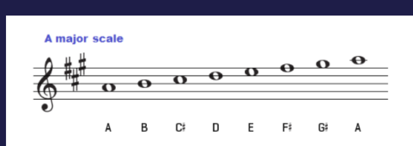
REFERENCES

1. Chuan, C. H., & Chew, E. (2007). A hybrid system for automatic generation of style-specific music.
2. Allan, M., & Williams, C. K. I. (2005). Harmonising chorales by probabilistic inference.
3. Wikipedia: Applied of Markov Chain in Music

Materials

Dataset

MIDI files of classical compositions (e.g., Bach's chorales).



Software

Python libraries such as MIDIUtil, librosa, FluidSynth, and pydub were used for generating and processing MIDI files and audio.
(Appendix B1)

Discussion

The project demonstrates the utility of Markov Chains in building algorithmic components for discovering musical structures. However, while first-order models effectively capture short-term dependencies, they struggle with long-term coherence. Higher-order models integrating deep learning can address this limitation. Additionally, style optimization by adjusting the transition matrix opens the possibility of creating songs. Our biggest challenge during the project was balancing randomness with coherence in melody generation.

Product by Group 17

Method

1

Define the State

Translate the sheet music to the matrix of notes *(Appendix A1, A2)*

2

Construct A Transition Matrix

Analyze a dataset of existing music (a corpus) to calculate the probabilities of transitions between states. (*Appendix A3)*

3

Generate Music *(Appendix B2)*

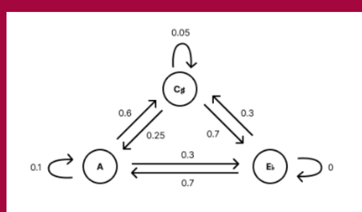
1. Start with an initial state.
2. Use the transition probabilities to select the next state randomly.
3. Repeat this process to create a sequence of notes, chords, or rhythms.

4

Post-Process and Enhance

Apply musical to refine the generated sequence. Converted the generated melody into MIDI format and exported it as an MP3 file. *(Appendix B3)*

Results



A melody with 20 notes was generated successfully. The generated song, stored in MIDI and MP3 formats, reflects the characteristics of the original melody while introducing variations. The output files demonstrate the effectiveness of Markov chains in modeling note transitions and durations. (*Appendix C)*

Abstract & Conclusion

The MAV project applies Markov Chains to analyze and compose classical music. By studying existing pieces, we construct transition matrices to model note and chord progressions, enabling the generation of new compositions that reflect historical stylistic traits. This research highlights the role of probabilistic modeling in classical music composition.

Member Contributions

SeanLin
Research Markov Chains Function

Rlong
Code Project base on research

Nhat Anh
Research about method and abstract

Thomas
Create slide and record video

Rong
Introduction to Markov Chains

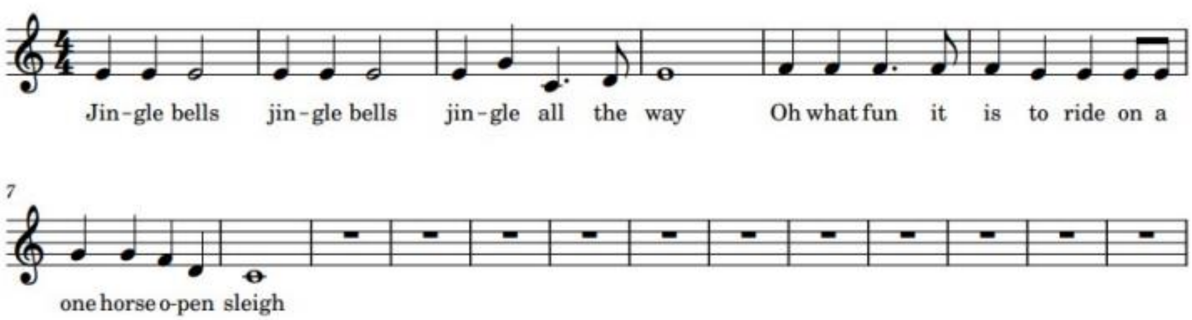
Justin
Create slide and record video

Product by Group 17

Appendix

A) Base

A1. Demo Sheet Music



A2. Translate the Sheet Music to Matrix

Note	Duration
E5	1
E5	1
E5	2
E5	1
E5	1
E5	2
E5	1
G5	1
C5	1
D5	1
E5	4

Note	Duration
F5	1
F5	1
F5	0.5
F5	0.5
F5	1
E5	1
E5	1
E5	1
G5	1
G5	1
E5	4

A3. Calculate the Matrix

MARKOV CHAINS AND MUSICAL COMPOSITION

	E4	E2	G4	C3	D8	E	F4	F3	F8	E8	D4	C
E4	0.43	0.29	0.14	0	0	0	0	0	0	0.14	0	0
E2	1	0	0	0	0	0	0	0	0	0	0	0
G4	0	0	0.5	0.5	0	0	0	0	0	0	0	0
C3	0	0	0	0	1	0	0	0	0	0	0	0
D8	0	0	0	0	0	1	0	0	0	0	0	0
E	0	0	0	0	0	0	1	0	0	0	0	0
F4	0.25	0	0	0	0	0	0.25	0.25	0	0	0.25	0
F3	0	0	0	0	0	0	0	0	1	0	0	0
F8	0	0	0	0	0	0	1	0	0	0	0	0
E8	0	0	0.5	0	0	0	0	0	0	0.5	0	0
D4	0	0	0	0	0	0	0	0	0	0	0	1
C	1	0	0	0	0	0	0	0	0	0	0	0

B) Code

Link Code: https://colab.research.google.com/drive/1rjM0OGcfZP4ZHo-2XLfeK7lmM1WMsaXN?usp=sharing

B1. Setup

First, we have to install python libraries (Assume OS: Ubuntu)

```
# install the library we need
!pip3 install midiutil
!sudo apt-get install fluidsynth
!pip3 install midi2audio
!pip3 install pydub
```

Second, we run the following code

```
# import the library we need
from midiutil import MIDIFile
import librosa
from collections import defaultdict
import random
from midi2audio import FluidSynth
from pydub import AudioSegment
import os

# this is the original song's note and note duration
# you can change to any song you like
original_melody = [
    ("G4", 0.5), ("E4", 0.5), ("E4", 1.0), ("F4", 0.5), ("D4", 0.5),
("D4", 1.0),
    ("C4", 0.5), ("D4", 0.5), ("E4", 0.5), ("F4", 0.5), ("G4", 0.5),
("G4", 0.5), ("G4", 1.0)
]
```

B2. Build markov chain for note and note duration

```
def build_markov_chain_for_notes_and_durations(melody):
    note_chain = defaultdict(lambda: defaultdict(int))
    duration_chain = defaultdict(lambda: defaultdict(int))

    for i in range(1, len(melody)):
        prev_note, prev_duration = melody[i-1]
        current_note, current_duration = melody[i]

        note_chain[prev_note][current_note] += 1
        duration_chain[prev_duration][current_duration] += 1

    for prev_note in note_chain:
        total = sum(note_chain[prev_note].values())
        for next_note in note_chain[prev_note]:
            note_chain[prev_note][next_note] /= total

    for prev_duration in duration_chain:
        total = sum(duration_chain[prev_duration].values())
        for next_duration in duration_chain[prev_duration]:
            duration_chain[prev_duration][next_duration] /= total

    return note_chain, duration_chain
```

```python
# I originally wanted to write something, but I found it to be very
intuitive. zzz.
# Just like what I say, compute the time of all the note to the note
and the time of its total apperance.

def generate_melody(note_chain, duration_chain, start_note,
start_duration, num_notes=10):
    # (for those who haven't learnd python) Unlike C/C++, python don't
need to claim a enough length for an array first.
    # It can directly use arr.append("the_thing_you_want_to_insert")
to append something.
    generated_melody = [(start_note, start_duration)]
    current_note = start_note
    current_duration = start_duration

    for _ in range(num_notes - 1):
        # According to the probability, random choice
        if current_note in note_chain:
            next_note = random.choices(
                list(note_chain[current_note].keys()),
                weights=list(note_chain[current_note].values()),
                k=1
            )[0]
        else:
            next_note = random.choice(list(note_chain.keys()))

        # According to the probability, random choice
        if current_duration in duration_chain:
            next_duration = random.choices(
                list(duration_chain[current_duration].keys()),
                weights=list(duration_chain[current_duration].values()
),
                k=1
            )[0]
        else:
            next_duration = random.choice(list(duration_chain.keys()))

        generated_melody.append((next_note, next_duration))
        # renew the next to current
        current_note = next_note
        current_duration = next_duration

    return generated_melody
```

B3. Output

```
# To tranfer to the .mp3
# This part is unimportant. You guys can ignore it.
def frequencies_to_midi(frequencies, output_file):
    midi = MIDIFile(1)
    track = 0
    time = 0
    midi.addTrackName(track, time, "Track")
    midi.addTempo(track, time, 120)

    for note, duration in frequencies:
        midi_note = int(librosa.hz_to_midi(librosa.note_to_hz(note)))
        midi_note = max(0, min(127, midi_note))
        midi.addNote(track, 0, midi_note, time, duration, 100)
        time += duration

    with open(output_file, 'wb') as outf:
        midi.writeFile(outf)


def midi_to_mp3(midi_file_path, mp3_file_path):
    fs = FluidSynth()
    wav_file_path = "temp_audio.wav"
    fs.midi_to_audio(midi_file_path, wav_file_path)

    audio = AudioSegment.from_wav(wav_file_path)
    audio.export(mp3_file_path, format="mp3")
    os.remove("temp_audio.wav")
    # os.remove(midi_file_path)

note_chain, duration_chain = build_markov_chain_for_notes_and_durations(original_melody)

# If you want to listen more, you can adjust the "num_notes" by yourself. It's about the number of note it generate.
generated_melody = generate_melody(note_chain, duration_chain, "G4", 0.5, num_notes=20)

frequencies_to_midi(generated_melody, "generated_song.mid")
```

C) Result

D) Student ID

113550022 SeanLin
113550031 Rlong
113550201 Nhat Anh
113550122 Thomas
113550118 Rong
113550063 Justin