Fire Factory: Automated Lofi Melody Generation Using ML/AI

Ansh Shukla
University of Victoria
ashukla@uvic.ca
V00816280

Jonathan Parkes
University of Victoria
jparkes@uvic.ca
V00826631

Eileen Eng
University of Victoria
esceng@uvic.ca
V00844647

Ryley Woodland
University of Victoria
ryleyw@uvic.ca
V00151045

ABSTRACT

Project Fire Factory is an automated lofi melody generator that incorporates Artificial Intelligence (AI) and Machine Learning (ML) techniques. We will be utilizing Python libraries such as Librosa, ADTLib to conduct calculations on a lofi data set which will isolate key features such as audio fingerprints, pitch, timbre, rhythm, and Mel-frequency cepstral coefficient (MFCC) data. These features can then be fed into a genetic algorithm that will loop through reproduction, mutation, recombination, and selection of promising outputs. The final output will be a ~3 minute lofi audio track produced by this algorithm.

The project will be divided into 4 sprints that span from October 19, 2020 to December 11, 2020. Some key tasks for this project are creating documentation, designing and making subsystems, testing, performing subsystem integration and preparing for the application demo.

1. INTRODUCTION

AI and ML have accelerated the growth of many fields. AI technology is now responsible for automating countless tasks that were historically arduous and repetitive for humans []. Humans have always excelled in creativity, but ML may be a serious contender in this area, specifically, song composition. By incorporating various AI techniques such as digital fingerprinting, MFCCs, K-means clustering, and genetic algorithms, we can produce automated lofi melodies. The project, Fire Factory, aims to automate the production of a dynamic audio track that is comparable to lofi songs engineered by humans.

2. GOALS

2.1 Input Data

Although Fire Factory is focused on producing lofi beats, the input data will consist of both lofi and non-lofi audio samples. Lofi music is loosely defined as being non-lyrical and incorporates elements from electronic music and underground instrumentals from New York during the 1960s and 1970s. Most songs from this music genre are relaxing and contain beats similar to hip hop.

The input data will be coming from 2 datasets, a lofi dataset for examples of lofi music, and a non lofi dataset containing bad outputs or music distinctly different from lofi. The importance of the second dataset is that it allows the algorithm to have a sense of what not to make. Each data sample will range from 5 seconds to 5 minutes, with a minimum of 500 input samples. The initial file formats in the data set (Figure 1) will be audio files such as mp3, where they are public domain music retrieved from sources like Youtube, Reddit, SoundCloud, etc.

2.2 Preprocessing

Before sending data into our algorithm, we must extract specific information from the lofi songs. The information required includes:

- Pitch
- Timbre
- Rhythm
- MFCC data
- Audio fingerprints

Pitch data will be extracted using zero-crossings on a time domain [13]. This pitch data will be used to generate melodies as our algorithm composes music. To identify the core melodies of songs, repeated melodies will be identified and extracted individually. The Python Librosa package provides libraries for extracting pitch and zero crossing data [8].

Timbre data will be extracted by categorizing the song's brightness (spectral centroid), decay (spectral roll-off), and magnitude fluctuation (spectral flux) using the corresponding equations [15]. The data will then be used to choose a type of instrument to play in the final audio signal. Librosa can also help extract each of these features [8].

Rhythm data will be extracted by tempo estimation, drum transcription, and pattern analysis [14]. This data will be used to categorize different types of rhythms that can be overlaid into our final audio signal. Identifying the rhythm of a song before the other categories will allow us to segment the song accurately resulting in full information in selected bars of music. Librosa has libraries for tempo estimation [8] while ADTLib will provide libraries for drum transcription [9]. Pattern analysis can be conducted by counting patterns as they are identified to measure frequency.

MFCC data will be generated in order to be used in our genetic algorithm (discussed in the next section) and also as an accuracy measurement for our generated audio [15]. Although MFCC's can be used to categorize the timbre of a sound, the quality will be lost when trying to convert an

MFCC back to a sound. The MFCCs will be used to "define" what lofi music is, and compare our generated audio to an exemplary MFCC vector. Librosa provides libraries for MFCC extraction [8].

Finally, a model audio fingerprint (spectral peak fingerprint) will be created from our sample lofi music to later compare to an audio fingerprint created from our algorithm's output. This will be done using NumPy arrays and some analysis algorithms on our part, using some sources such as *Audio Fingerprinting with Python and Numpy [10]*.

All of these features will be represented as NumPy arrays and stored in a local database for use in the algorithm.

2.3 Genetic Algorithm

When creating an algorithm that is capable of music composition, we must first consider how music is composed. Additionally, the algorithm must be able to make unique music each time; meaning it can't follow a linear process. This is why genetic algorithms are the ideal choice for this project. A genetic algorithm "... is a metaheuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms (EA)." [6]. The ability for this algorithm to act in a somewhat random but evolutionary way makes it the best choice for automatic music composition [4], [5], [12]. The key components of our genetic algorithm will be melody generation, melody rating, melody splicing, and an exit condition.

Melody generation is the first step of our algorithm and we will start by creating 6 melodies. Although a larger population is more effective as shown in *Optimal Population Size and the Genetic Algorithm* [7], our initial population size will be smaller for development and time purposes. This process will only happen once, and the results will be referred to as Generation 1.

Melody rating (our fitness function) is the next step of our algorithm. It will use multiple heuristics to give a rating for each of the melodies. The heuristics we plan to use will compare digital fingerprints, and compute a generalized MFCC[11] of the lofi songs. This is a very important step in the fitness function to do the mutation process (splicing stage).

Splicing is used to create the next generation of melodies through a process inspired by mutation and natural selection in nature. To do this, the top-rated melodies are then split and combined in various ways to regenerate to the original population size of 6. This process will slowly be mixing and matching only the best of each generation, leading to the best melody being created.

The final part of this algorithm requires an exit condition to be defined. Since the algorithm works similarly to natural selection, there are a few ways to define an exit condition. The first way is to define a set number of generations, that when reached, will output the best melody. One benefit of this method is that the running time will be predictable and can be adjusted accordingly. On the other hand, this method does not guarantee good results every run.

The second way is to define an exit condition based on the top rating. This means that it will stop when the rating of a melody has reached or exceeded a specific value. This is generally going to give more consistent quality but a varying runtime. We will incorporate both techniques throughout the project focusing more on method 1 during the development process, and method 2 during the final algorithm.

2.4 Output Melody

The produced audio track will be approximately 3-minutes in length. To ensure that the melody is acceptable, outputs will be scored against the original input data as well as listened to by human ears. In the case that the output is unsatisfactory, it will be placed into the non-lofi data to allow the algorithm to deviate away from outputs similar to those.

3. PLAN

The proposed structure of the machine subsystem is shown in Figure 1.

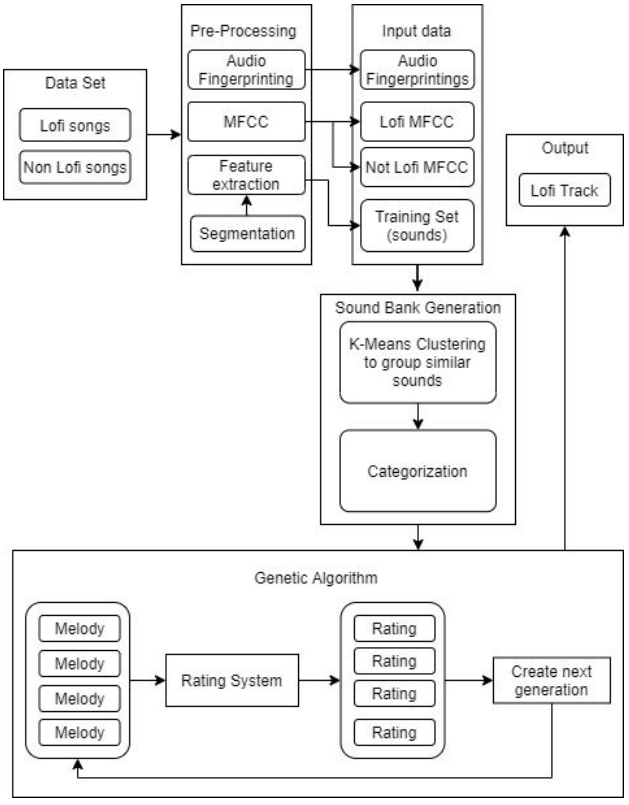

Figure 1. Model Workflow

There will be a custom Python API that will interact with the model to retrieve the output and present it to a user in some fashion (media streaming, download, etc).

3.1 TASK BREAKDOWN

3.1.1 Sprint 1

Sprint 1 goes from October 19th to October 30th and will be focusing on the initial project setup. We plan on using this sprint to identify which features we want to extract from audio samples for the lofi genre. We will then prepare the two data sets; one for lofi songs, and the other for non-lofi songs. These two sets will be used to train our model as shown in Figure 1. We plan on mapping out that system interaction (Figure 1) to initial software architecture as skeleton code. To do this, we plan on making basic prototypes of each subsystem in Python and also create reference documentation. Additionally, we will allocate some of this sprint on the report deliverable. Most of this sprint will be completed as a group as it requires inputs from all members, and for all members to understand the architecture of the system.

3.1.2 Sprint 2

Sprint 2 goes from October 31st to November 13th and will focus more on the implementation of Fire Factory. The goals for this sprint will include the development of the feature extraction, MFCC vectors, audio fingerprinting, clustering of sounds, and the genetic algorithm. The issues and project board on GitHub have already been created to begin our sprint. Issues will be divided among the members as the sprint progresses, however our team lead will delegate work if needed. All members of this group communicate well and are self-motivated so disproportionate work distribution is not an issue. To ensure that every member knows how each implementation works, we will require multiple peer reviews on GitHub to merge a pull request. Sprint Reports will also be done collaboratively.

3.1.3 Sprint 3

Sprint 3 will run from November 14th to November 27th and will focus on subsystem integration and testing. Since several subsystems will be developed in parallel, we will need to make sure the entire system works together perfectly. We expect to encounter some bugs, so we have allocated a whole sprint for testing and bug fixes as they arise. We also plan to integrate the model and the API together during this sprint. Most of the work during this sprint will have to be done collaboratively since we will need to communicate and explain our implementations to each other. Like previous sprints, the sprint report will be done as a group.

3.1.4 Sprint 4

Sprint 4 spans from November 28th to December 11th. We plan on using this sprint for final touch-ups and to potentially set up a server for our application. If the application is hosted, we are aiming to demo our prototype in front of our peers. The rest of the sprint will be spent preparing for presentations and the final report.

4. REFERENCES

[1] Data Council. "Music Information Retrieval using Scikit-learn (MIR algorithms in Python) - Steve Tjoa," YouTube, November 10, 2014. Available: https://www.youtube.com/watch?v=oGGVvTgHMHw [Accessed October 5, 2020]

[2] Colombo, F. et al. "Algorithmic Composition of Melodies with Deep Recurrent Neural Networks," arXivLabs. https://arxiv.org/abs/1606.07251 [Accessed October 15, 2020].

[3] G. Seewooruttun, "Transforming By Automating Intelligently With Artificial Intelligence (AI) Is Now The Practice Of Early Adopters With An Eye To The Future," Ernst & Young Global. https://www.ey.com/en_gl/consulting/how-ai-is-automating-intelligently [Accessed October 16, 2020].

[4] N. Tokui, H. Iba, "Music Composition with Interactive Evolutionary Computation," Iba Lab. http://www.iba.t.u-tokyo.ac.jp/papers/2000/tokui-GA2K.pdf [Accessed October 16, 2020].

[5] W. Roetzel et al, "Chapter 6 - Optimal design of heat exchanger networks," ScienceDirect. https://www.sciencedirect.com/science/article/pii/B9780128178942000066 [Accessed October 14, 2020].

[6] Eiben, A. E. et al. "Genetic algorithms with multi-parent recombination". PPSN III: Proceedings of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature: 78–87. ISBN 3-540-58484-6.

[7] Gotchall, S., Rylander, B., "Optimal Population Size and the Genetic Algorithm," http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.105.2431&rep=rep1&type=pdf [Accessed October 14, 2020].

[8] McFee, Brian, Colin Raffel, Dawen Liang, Daniel PW Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. "librosa: Audio and music signal analysis in python." In Proceedings of the 14th python in science conference, pp. 18-25. 2015.

[9] ADTLib Documentation. https://github.com/CarlSouthall/ADTLib. Accessed 17-10-2020.

[10] Drevo, W. "Audio Fingerprinting with Python and Numpy," Willdrevo. https://willdrevo.com/fingerprinting-and-audio-recognition-with-python/ [Accessed October 17, 2020].

[11] L. Rowe et al, A Machine Learning Approach to Emotion Classification," Ryleyw. https://www.ryleyw.com/static/media/seng474report.69aac467.pdf [Accessed October 17, 2020].

[12] J. Biles, "GenJam: A Genetic Algorithm for Generating Jazz Solos", Semanticscholar. https://www.semanticscholar.org/paper/GenJam%3

A-A-Genetic-Algorithm-for-Generating-Jazz-Biles/0d8bb76012371d534ae8fcfb779e570ea851c487. [Accessed October 18, 2020].

[13] Tzanetakis, G. "Time Domain Pitch Extraction Using Zero-Crossings." Kadenze. https://www.kadenze.com/courses/extracting-information-from-music-signals/sessions/monophonic-pitch-detection.

[14] Tzanetakis, G. "Tempo Estimation", "Drum Transcription and Pattern Analysis". Kadenze. https://www.kadenze.com/courses/extracting-information-from-music-signals/sessions/rhythm-analysis.

[15] Tzanetakis, G. "Spectral Descriptors and MFCCs." Kadenze. https://www.kadenze.com/courses/extracting-information-from-music-signals/sessions/audio-feature-extraction.