Project Milestone 3

Rohan Sharma, Kshitij Alwadhi, Siddhant Sharma

November 18, 2021

Contents

Abstract

In this project we propose and implement a MIDI based approach for music recognition. Traditional music recogniton systems like Shazam and DejaVu rely on spectrogram analysis, preventing the matching of song derivatives (covers, humming etc.). We propose a system that identifies, normalises and matches the melody from a song sample. We use the skyline algorithm for generating the melody of the songs and then proceeding to the retrieval task.

1 Introduction

Music recognition systems like Shazam and DejaVu rely on spectrogram analysis of samples to match songs. A major flaw in these algorithms is that they assume that the query given to the system has features (like sampling frequency) precisely similar to the studio recorded versions. Thus, matching song derivatives(like instrumental covers) is not viable.
To enable matching song derivatives, we need to find a *feature* common to all derivatives of a song[12]. The premise of our project is that this *feature* is the melody of the song. Think of happy birthday - Any version of the song will have the same familiar "Happy birthday" tune underneath it. If we could extract this melody consistently, perhaps we could arrive at a more general music recognition system. There are thus two problems we need to address :

1. Identifying the melody of a song excerpt
2. Identifying the original song based on this melody

2 Method

2.1 Melody Extraction

1. **Sample To Midi**-First the given audio sample is converted to the Midi format. This is done because MIDI records the audio's notes at any given time, giving a **standardized** interpretation of the song. We can thus treat it as a document and use retrieval models like the Vector Space Model.
2. **Standardisation**-The midi file is converted to the standard C major scale and note length normalisation.
3. **Melody Extraction**-Using the midi obtained above, we use the **skyline** algorithm to extract the melody from the file
4. **Melody to Midi**-Store the melody as a midi file

A schematic for melody extraction is given below

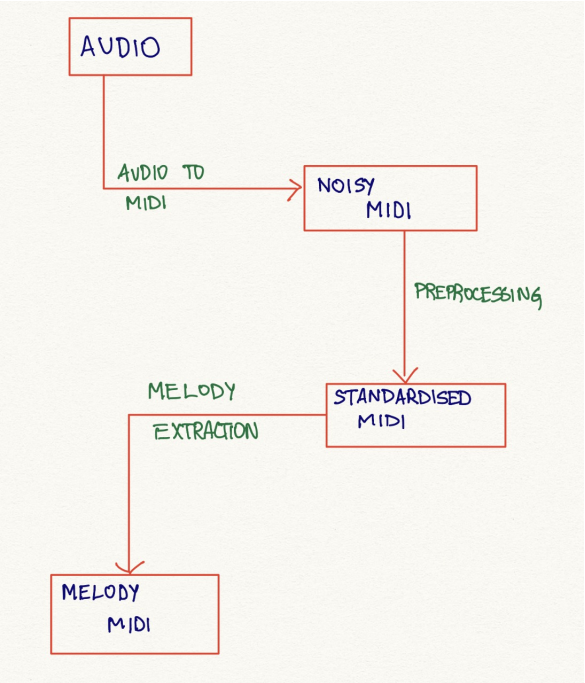

Figure 1: Pipeline of the project

2.2 Retrieval

1. **Sample to Midi and Standardisation**-The sample is converted to Midi using the same process as given above
2. **Melody Extraction**-Using the above midi file, we extract the melody of the song.
3. **VSM Retrieval**-Using the vector space model developed below we compute similarities with songs in the collection

3 Technical Details

Our project can be divided into a three step project where we first process our given songs database and convert them to MIDI files. Next we extract melody from the songs and store that as our database. Finally, we perform our retrieval task to match our queries (5 second song snippets) to the songs.

3.1 Data Pre-Processing

3.1.1 Dataset used

We will be using the infamous MAESTRO [5] dataset for our project. It contains roughly 200 hours of paired audio and their MIDI files from ten years of International Piano-e-Competition.

3.2 Melody Extraction

After the data is pre-processed and we have the MIDI files containing the cleaned up notes corresponding to monophonic audio, we propose to extract the melody of this audio.
The reason for this extraction is because we wish to find features of the audio that are unique to it and also are completely descriptive. As we know, all songs have a distinct melody (else they will get copyrighted :P) and the melody is enough from any intelligent system to determine the song.
Hence extracting the melody before modeling our audio is the correct and time efficient thing to do. Extracting the melody lowers the file size while also removing the noise from the audio.

3.2.1 Skyline Algorithm

The skyline algorithm [1, 9] is a novel state-of-the-art method for detecting melodies in an audio file and has not been explored and bench marked enough yet. More specifically, it has never been used on music retrieval tasks and hence its efficiency is unknown for this important and practical use.
The skyline algorithm primarily works on the principal that the highest note/pitch is the note that constitutes the melody. This is based on the fact that the human ears tend to pick up on the higher frequencies and hence using these notes as the most prominent ones (which constitute the melody) is a very valid assumption. Hence we pick the channel with the highest frequency note playing and silence all others, to sample out the melody.
A slight drawback [7] is that the Skyline algorithm assumes that the melody appears on one and only one channel with the highest note. Depending on the data, this can be a cause for failure, since the melody can be expressed by different instruments in different channels at different times. However, this failure will be constant amongst both queries and songs and despite the algorithm not being ideal, it is estimated that the Information Retrieval will be robust to this error!

```
import copy
from miditoolkit.midi.containers import Instrument
from miditoolkit.midi.parser import MidiFile

def skyline(filename, instr_idx=0):
    '''
    Input:
        filename: location of MIDI file
        instr_idx: index of instrument to extract melody from (piano is 0)

    Return:
        midi_melody: MidiFile
    '''
    midi = MidiFile(filename)
    notes = sorted(midi.instruments[instr_idx].notes, key = lambda x: x.start)
    skyline_notes = []
    finished_notes = []
    for si, note in enumerate(notes):
        if note in finished_notes:
            continue
        finished_notes.append(note)
        start_t = note.start
        notes_at_t = [note]
        pitches = [note.pitch]
        maxpitch = pitches[0]
        ind = 0
        if si != len(notes)-1:
            while start_t==notes[si+1].start:
                notes_at_t.append(notes[si+1])
                finished_notes.append(notes[si+1])
                pitches.append(notes[si+1].pitch)
                if maxpitch<pitches[-1]:
                    ind = len(pitches)-1
                    maxpitch = pitches[-1]
                si += 1
                if si == len(notes)-1:
                    break
        n = copy.deepcopy(notes_at_t[ind])
        if si < len(notes)-1:
            note.end = min(n.end, notes[si+1].start)
        skyline_notes.append(n)
    midi_melody = MidiFile()
    midi_melody.markers = midi.markers
    midi_melody.tempo_changes = midi.tempo_changes
    piano_track = Instrument(0, is_drum=False, name='piano')
    piano_track.notes = skyline_notes
    midi_melody.instruments = [piano_track]
    return midi_melody

midi_final = skyline('test.mid')
midi_final.dump('test_melody.mid')

```

3.3 Music Information Retrieval

There exists a few MIR techniques [3, 8, 11] but none of them use the techniques as detailed below. The information that we will be storing of our documents(songs), will be a list of tuples where the tuple will be of the form (note/pitch, start time, end time, velocity). We will be storing the dataset by creating MIDI files using 5 second clips of the songs. A frames of 5 seconds will be moved along the songs in strides of 1 second. Similarly, our 5 seconds query will also be a list of these tuples.

3.3.1 V.S.M. Approach

What we plan on doing here is that we will be creating a vector space model where the indices will be decided by the pitch instead of the words in vocabulary and the weights associated with each index will be a basis function of the variables time duration and the velocity of the pitch,

$$V_j = \phi(vel, s, e)$$

$$Sim_{cosine}(m, q) = \frac{\vec{m}.\vec{q}}{\|\vec{m}\|\|\vec{q}\|}$$

here, j represents the pitch index, vel represents the velocity of the pitch and s represents the start time and e represents the end time of that particular pitch. The pitch can lie between (-8192, 8191) so we will have a 2*8192 dimensional vector for each song snippet representation with the weights corresponding to some basis function of the velocity and the time duration while that pitch was on during the complete audio.
We'll be comparing the performance between two settings. One where we will be creating these vectors from the notes present in the original song and then compare this with creating vectors from the notes extracted by the melody.
This process will have a major con as we will be disregarding the order of occurrence of the notes in the given 5 second clips however experimenting how this will the results will be like will give an indication of the performance of our two experiments, one with the vector space similarity model and then the effectiveness of using melody instead of the whole song.

3.3.2 Extension to Dejavu[10]

The approach followed in methods like Dejavu is that they create a spectrogram of the original song by using FFT over small windows. Once they have spectrogram, they proceed to the next step of finding peeks (local maxima of amplitudes) in that spectrogram. Now, the data left to store is the position of these maximas (only frequency and time required). These discrete (time,frequency) pairs are in theory resistant to noise. Since there can be an overlap between these peaks in different songs, they combine these peaks into fingerprints[2]. To create these fingerprints, they store these peaks parametrized by its neighbours

(time difference between nearest k peaks) by passing this information to a hash function and storing the output (SHA1). The value of k (termed as Fan Value), is a hyper-parameter and controls how many peaks should a particular peak be associated with before passing to the hash function. A higher fan value will lead to greater number of fingerprints but also better accuracy.

What we plan on experimenting is to use this extracted melody for creating the fingerprints for Dejavu and similarly feeding the melody extracted from queries for the retrieval part. The expectation is that the retrieval speed will be better since the number of fingerprints will be way lesser in quantity.

4 Preliminary Results

We have successfully implemented the first part of our pipeline. Our system can convert a file to MIDI, process it and extract the melody. A detailed illustration of this process is below. We use the piano version of the **Happy Birthday** song.

1. **Sample To Midi**-First the happy birthday audio sample is converted to the Midi format. A visual representation [4] of the file is given below

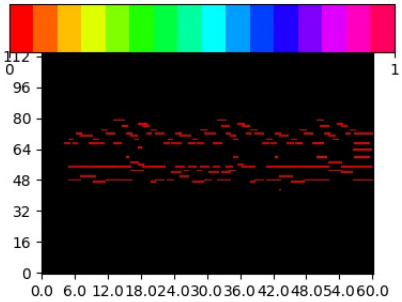

Figure 2: Original Song

2. **Melody Extraction**-Using the above midi file we extract the melody

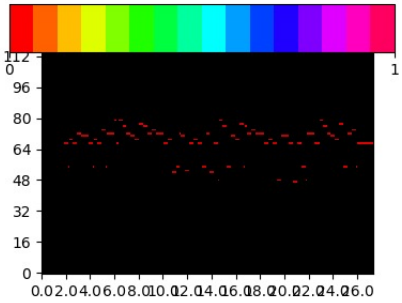

Figure 3: Extracted melody of the song

As can be seen from the two figures, the extracted melody contains substantially fewer notes.

We have also uploaded the original and the extracted melody .wav files of the Happy Birthday song: link to original and link to melody.

5 Remaining Work

The following tasks remain in our project

1. **Standardisation**-We have yet to standardise the midi files we get. The exact approach to. this hasn't been decided yet but will include scale and length normalisation

2. **Retrieval Model**-The second part of the project-computing melodic similarities between two midi files-still remains to be completed. However, we have started working on it.

3. **Representations of Melody**: We also plan on exploring the Melody2Vec[6] algorithm for creating the vectors.

6 Possible Applications

1. **Music Recognition**-The most obvious application of our project is in music recognition. The user can input a song derivative as their query and get back the song from which the sample was excerpted-thus working as an extended version of Shazam

2. **Melodic Similarity**-Suppose a musician writes a melody. He could send this melody as a query and get back a list of results of pre-existing songs that sound similar. He could then use ideas from these songs to continue his song.

3. **Copyright Infringement Detection**-Since the project is based essentially on melody matching, it could also be used on platforms like Youtube where detecting whether a video has used a copyrighted song becomes important

References

[1] Melody extraction.

[2] Sungkyun Chang. Neural audio fingerprint for high-specific audio retrieval based on contrastive learning. original-date: 2021-02-10T07:16:19Z.

[3] Vikram Dara and Shashi Mogalla. Pattern based melody matching approach to music information retrieval. 4(6):78–78. Number: 6.

[4] exeex. *midi-visualization*. Oct 2021.

[5] Curtis Hawthorne, Andriy Stasyuk, Adam Roberts, Ian Simon, Cheng-Zhi Anna Huang, Sander Dieleman, Erich Elsen, Jesse Engel, and Douglas Eck. Enabling factorized piano music modeling and generation with the MAESTRO dataset. In *International Conference on Learning Representations*, 2019.

[6] Tatsunori Hirai and Shun Sawada. Melody2vec: Distributed representations of melodic phrases based on melody segmentation. *Journal of Information Processing*, 27:278–286, 2019.

[7] Zheng Jiang and Roger B Dannenberg. Melody identification in standard MIDI files. page 7.

[8] Rainer Typke. Music retrieval based on melodic similarity.

[9] Alexandra Uitdenbogerd and Justin Zobel. Melodic matching techniques for large music databases. In *Proceedings of the Seventh ACM International Conference on Multimedia (Part 1)*, MULTIMEDIA '99, page 57–66, New York, NY, USA, 1999. Association for Computing Machinery.

[10] worldveil. Dejavu. `https://github.com/worldveil/dejavu`, 2019.

[11] Yoyo. Music transcription with semantic model. original-date: 2018-10-30T04:41:52Z.

[12] Yi Yu, Roger Zimmermann, Ye Wang, and Vincent Oria. Recognition and summarization of chord progressions and their application to music information retrieval. In *2012 IEEE International Symposium on Multimedia*, pages 9–16.