Kaz Erdos

Karthik Seetharaman

CS Project Proposal

11/16/20

Statement of Goals

This product does not aim to solve an existing problem, but rather to broaden the scope of classification technology by extending it to classical music. Specifically, the main goal of the program is to be able to identify the style of composition of an inputted piece of classical, keyboard music. This will be done through the use of machine learning, where the computer learns the characteristics of different periods of classical keyboard music (Baroque, Classical, Romantic, etc). Possible extensions are to identify the specific composer of a piece from a designated list, and to mimic the composing style of a particular composer. This application is intended for anyone with an interest in classical music, including students, teachers, composers, performers, and so on.

Functional Description - Minimal Viable Product

The application should have the following minimal features:

1) The program should be able to classify the style of composition of an inputted piece of classical keyboard music with reasonable accuracy. These styles include Baroque, Classical, Romantic, Post-Romantic, Impressionistic, and Modern.
2) The program should be able to convert audio to MIDI files to condensed text (converting audio to MIDI files to text is a necessity and can be done quite easily through the use of online programs. However, condensing the text will require some more programming).
3) If applicable, the application should be able to classify the composer from a limited list of composers.
4) (extension) The program should be able to produce a MIDI file that accurately mimics the composing style of some prespecified composer (say, Bach).

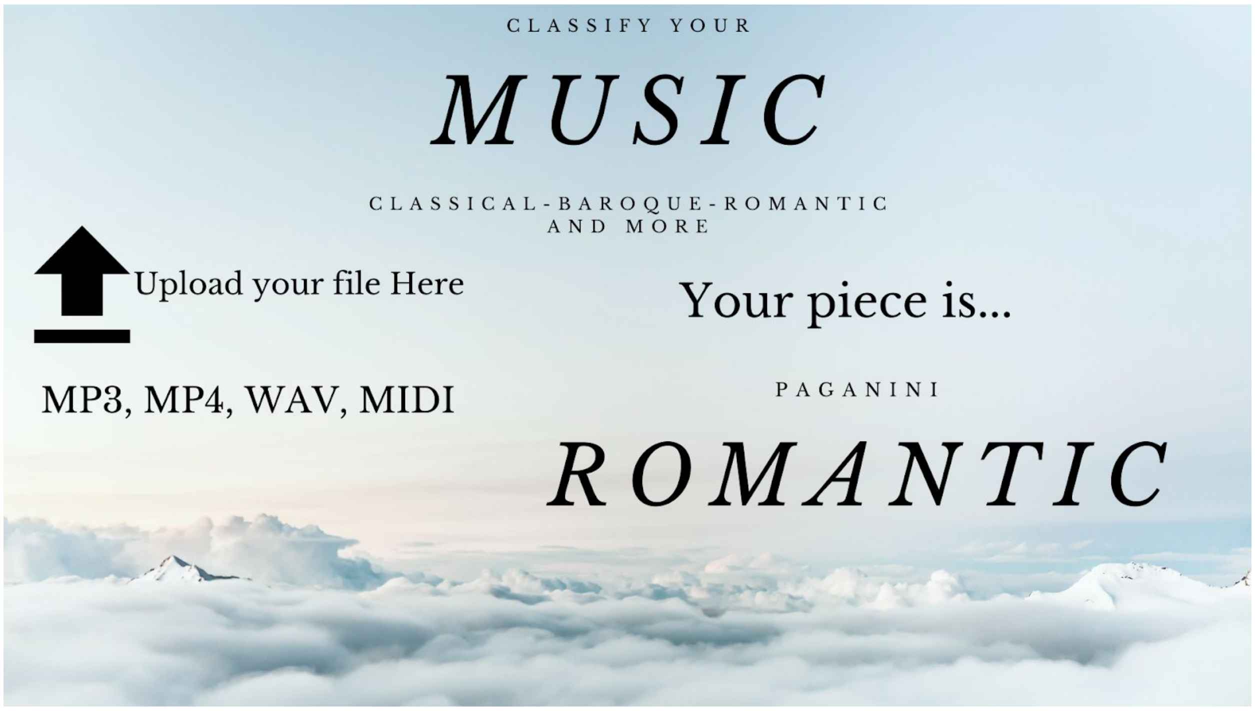
Technical and Data Feasibility

In its simplest form, the application takes in MIDI files converted to text as input. This is done through the MIDICSV convertor found at http://www.fourmilab.ch/webtools/midicsv/. A MIDI file is represented in text through a series of lines, each of which characterizes a note via the time it is played, its volume, and its pitch. This text file will then be condensed via a standard

Python program (the particular language does not matter here) and used as a piece of training data for the application. MIDIs for various pieces of classical music can be easily found on the internet. For example, here is a website containing many MIDIs of Bach pieces: http://www.jsbach.net/midi/. Future iterations of the model aim to input sound directly into the application. Audio can be converted to MIDI using many softwares, some of which are outlined here: https://www.dawsons.co.uk/blog/how-to-convert-audio-midi.

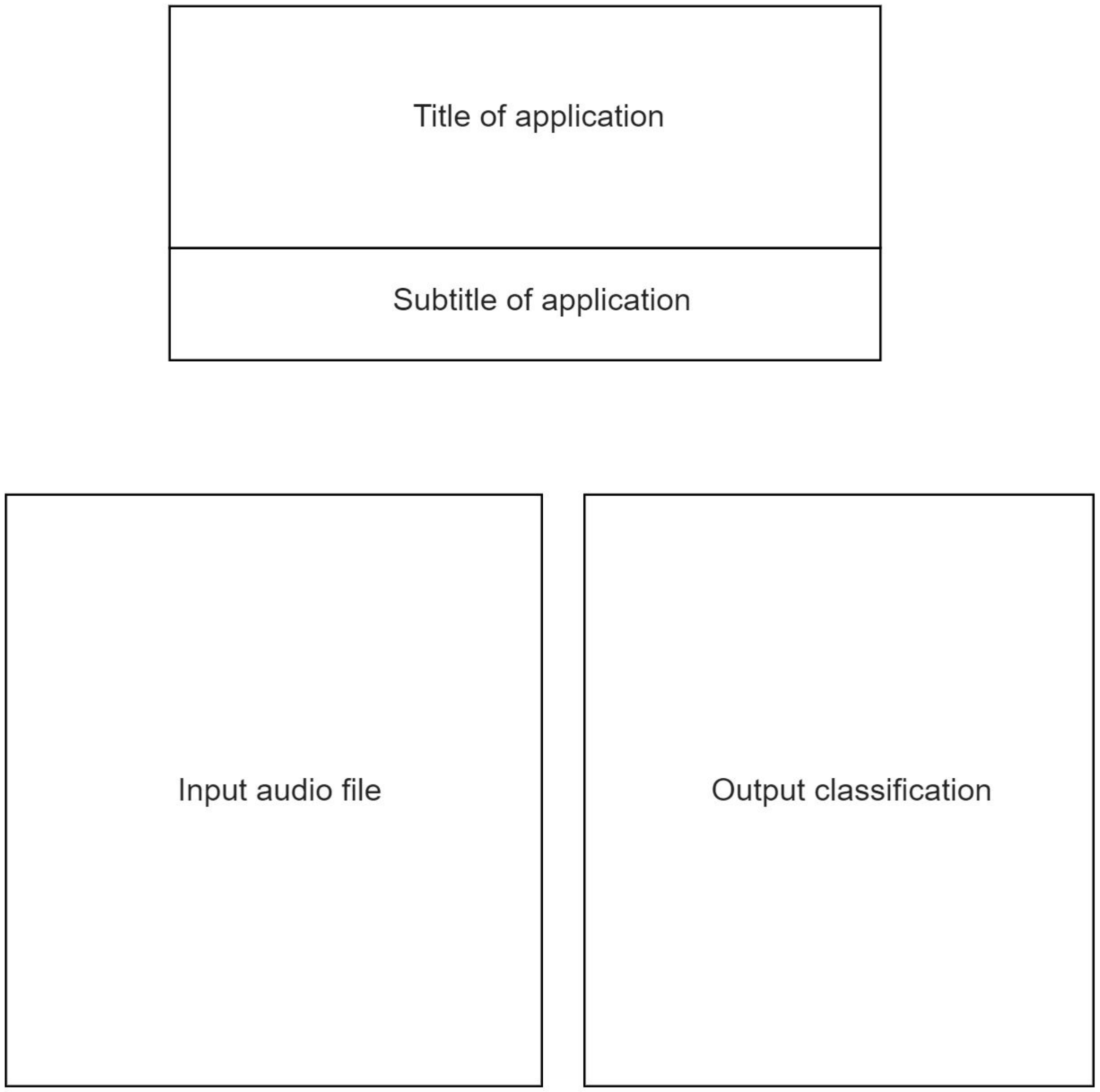
In terms of the actual machine learning software, the application will use TensorFlow, an open-source machine learning concept made by Google. We chose TensorFlow for its ease of use, and also its ability to be worked on collaboratively (Google Colab). The model will be trained with a database of condensed text files converted from MIDIs from each major period mentioned earlier. Unfortunately, TensorFlow runs only on NVIDIA graphics cards, though we have the required means available at home (NVIDIA GTX 1070). Once the model is trained, it can be transferred to the internet following an example shown here which utilizes Python and some Javascript:
https://towardsdatascience.com/how-to-deploy-tensorflow-models-to-the-web-81da150f87f7

User Interface

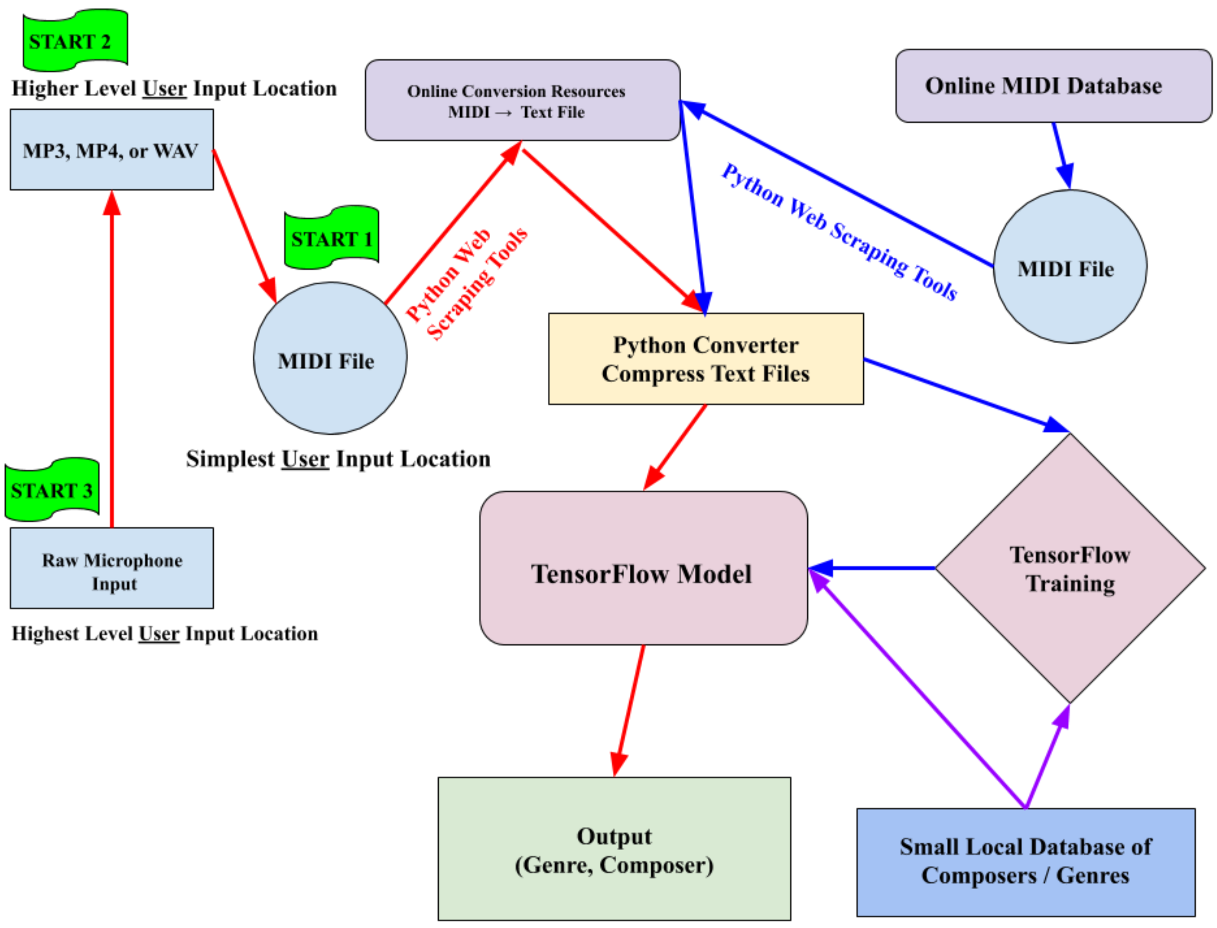


Shown above is a simple example of the user interface. As most of the work is done behind the scene, all that is needed for the first implementation of the project is a place to upload the files and a place to display the results. An example result of the model is presented, outputting “Romantic” as the genre and “Paganini” as the composer. In the future perhaps this interface can be adapted to include smart recommendations for pieces, as well as the aforementioned mimicry component. The website should be able to accept all 4 file formats listed, those being MP3, MP4, WAV, and MIDI.

Below is a simple wireframe of the proposed user interface:


Title of application
Subtitle of application
Input audio file
Output classification

Flow Chart


START 2
Higher Level User Input Location
MP3, MP4, or WAV
START 1
MIDI File
Simplest User Input Location
START 3
Raw Microphone Input
Highest Level User Input Location
Online Conversion Resources MIDI → Text File
Python Web Scraping Tools
Online MIDI Database
Python Web Scraping Tools
MIDI File
Python Converter Compress Text Files
TensorFlow Model
TensorFlow Training
Output (Genre, Composer)
Small Local Database of Composers / Genres

Shown on the flowchart are the three possible locations for user input, numbered START 1-3. The flowchart outlines how the data flows from the user into the model, as well as how the model is trained with data. At this point in time the flowchart does not include the mimicry extension, for obvious reasons.

Data Storage

After it is trained, the application's classification mechanism will not require any substantial database of data to pull from; the only data it will use is the inputted data, which will not be stored after the program is completed. Technically, the program needs access to the list of 5-6 styles of composition as well as a small list of composers for choices of classification, but this is an extremely small database that can be stored directly in the application. Even in the case of mimicking a composer, the computer will be able to draw off its memory from training and not require any data except the given input. The training data itself can be easily stored locally as text files, as even text files representing long pieces do not take up much space. If necessary, these files can also be moved to an online repository, such as GitHub.

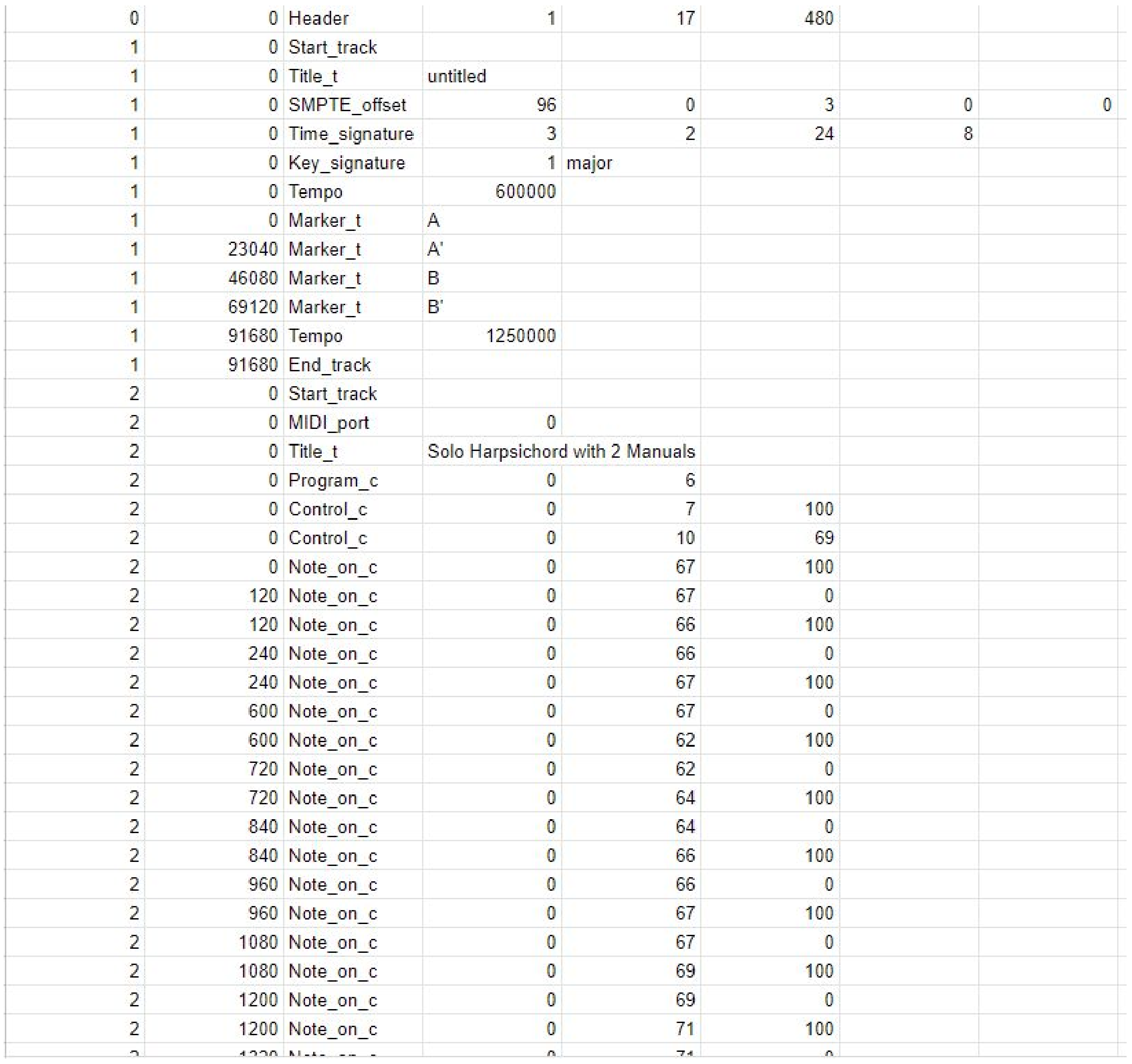
Python Scripts for Downloading and Converting Data

```python
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from pathlib import Path
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.common.by import By
import time
import os
chromeOptions = webdriver.ChromeOptions()
prefs = {"download.default_directory" : "C:\\Users\\Kazuya\\webscraping\\Beethoven\\MIDIs"}
chromeOptions.add_experimental_option("prefs",prefs)
chromedriver = "C:\\Users\\Kazuya\\Downloads\\chromedriver_win32\\chromedriver.exe"
driver = webdriver.Chrome(executable_path=chromedriver, options=chromeOptions)



def scrapeWebsite():
    driver.get('http://www.piano-midi.de/beeth.htm')
    time.sleep(1)
    elements = driver.find_elements_by_xpath("//a[contains(@href, '.mid')]")
    for x in range(0,len(elements)):
        if elements[x].is_displayed():
            elements[x].click()



def convertFiles():
    arr_txt = [x for x in os.listdir('C:\\Users\\Kazuya\\webscraping\\Beethoven\\MIDIs') if x.endswith(".mid")]
    print(arr_txt)
    for x in arr_txt:
        p="C:\\Users\\Kazuya\\webscraping\\Beethoven\\MIDIs\\"+x[:-4]+".bat"
        myBat = open(p,'w+')
        c = "C:\\Users\\Kazuya\\Desktop\\midicsv-1.1\\Midicsv.exe "+x+" "+x[:-4]+".csv"
        myBat.write(c)
        myBat.close()

scrapeWebsite()
convertFiles()
```

```python
import csv
import os
name = ['bach']
arr_txt = [x for x in os.listdir('C:\\Users\\Kazuya\\webscraping\\Bach\\MIDIs') if x.endswith(".csv")]
print(arr_txt)
for x in arr_txt:
    outputName = "COM_"+x
    with open(x, 'r') as inp, open(outputName, 'w', newline='') as out:
        writer = csv.writer(out)
        writer.writerow(name)
        for row in csv.reader(inp):
            if str(row[2]) == " Note_on_c":
                writer.writerow((row[0],str(row[1])[1:], str(row[4])[1:], str(row[5])[1:]))
```

Example CSV File (Converted from MIDI)

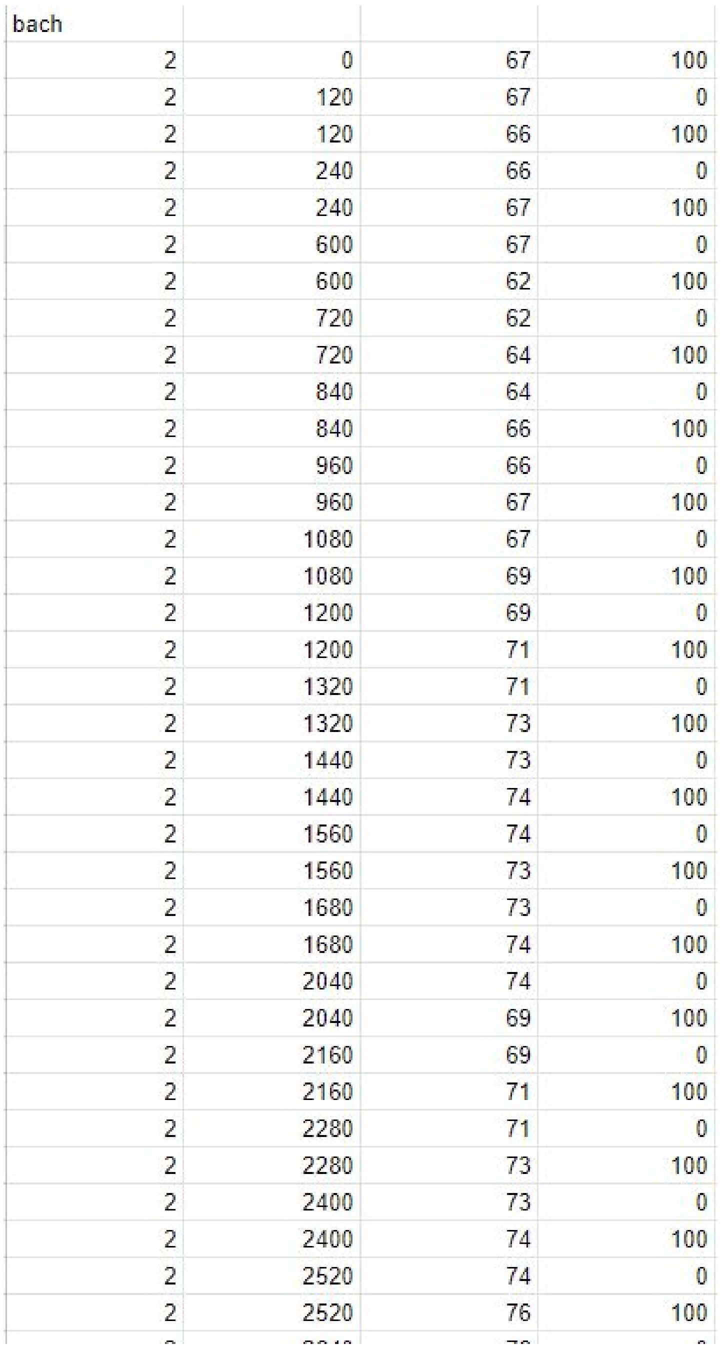
Before Compression:



0	0	Header	1	17	480		
1	0	Start_track					
1	0	Title_t	untitled				
1	0	SMPTE_offset	96	0	3	0	0
1	0	Time_signature	3	2	24	8	
1	0	Key_signature	1	major			
1	0	Tempo	600000				
1	0	Marker_t	A				
1	23040	Marker_t	A'				
1	46080	Marker_t	B				
1	69120	Marker_t	B'				
1	91680	Tempo	1250000				
1	91680	End_track					
2	0	Start_track					
2	0	MIDI_port	0				
2	0	Title_t	Solo Harpsichord with 2 Manuals				
2	0	Program_c	0	6			
2	0	Control_c	0	7	100		
2	0	Control_c	0	10	69		
2	0	Note_on_c	0	67	100		
2	120	Note_on_c	0	67	0		
2	120	Note_on_c	0	66	100		
2	240	Note_on_c	0	66	0		
2	240	Note_on_c	0	67	100		
2	600	Note_on_c	0	67	0		
2	600	Note_on_c	0	62	100		
2	720	Note_on_c	0	62	0		
2	720	Note_on_c	0	64	100		
2	840	Note_on_c	0	64	0		
2	840	Note_on_c	0	66	100		
2	960	Note_on_c	0	66	0		
2	960	Note_on_c	0	67	100		
2	1080	Note_on_c	0	67	0		
2	1080	Note_on_c	0	69	100		
2	1200	Note_on_c	0	69	0		
2	1200	Note_on_c	0	71	100		
2	1320	Note_on_c	0	71	0		

The raw data after conversion has unnecessary information like the track name, key signature, and markers. Phrases like “Note_on_c” are not needed for machine learning.

After Compression:



bach			
2	0	67	100
2	120	67	0
2	120	66	100
2	240	66	0
2	240	67	100
2	600	67	0
2	600	62	100
2	720	62	0
2	720	64	100
2	840	64	0
2	840	66	100
2	960	66	0
2	960	67	100
2	1080	67	0
2	1080	69	100
2	1200	69	0
2	1200	71	100
2	1320	71	0
2	1320	73	100
2	1440	73	0
2	1440	74	100
2	1560	74	0
2	1560	73	100
2	1680	73	0
2	1680	74	100
2	2040	74	0
2	2040	69	100
2	2160	69	0
2	2160	71	100
2	2280	71	0
2	2280	73	100
2	2400	73	0
2	2400	74	100
2	2520	74	0
2	2520	76	100

After the compression only the composer name and numerical information about the notes are included (time, pitch, volume). This data, once partitioned, can be used for training the machine learning model.