WOO FACTOR MUSIC PROJECT DOCUMENTATION

Howton, J., Kim, J., Osmon, L., Tsegaselassie, W.

Table of Contents

Introduction

Our team has joined this project to help with early-stage research into how to understand and then automatically generate the basic architecture of a song, which is called its "chord progression.” Our client wants to use these to create the next generation of musical instruments.

For this project, we worked on developing a digital representation scheme that would encode all important information from a chart at the grammatical chord progression level. Based on that chosen representation scheme, we built a model that is able to predict possible next chords given an input progression.

Since we are working on an early stage of this exploratory project, documenting our process is important for smooth transitions and allowing future teams to easily follow along with what has been done so far. The general understanding of the project will allow future teams to have a clear notion of what to accomplish next and provide a quality product to the client.

In this document, we will explain what we chose as our digital representation scheme, how that was implemented in Python, documentation of the algorithms we used, a process flow chart, and a user guide.

Digital Representation Scheme

The first part of our project was coming up with a digital representation scheme that encoded all the relevant information we could get from the charts. To do this, the first step was to figure out exactly what was important to get from the charts and learn how to recognize that information.

Extracting Information from Charts

I. The two starting parts of information required from a chart are the key that the song is in and the chord progression

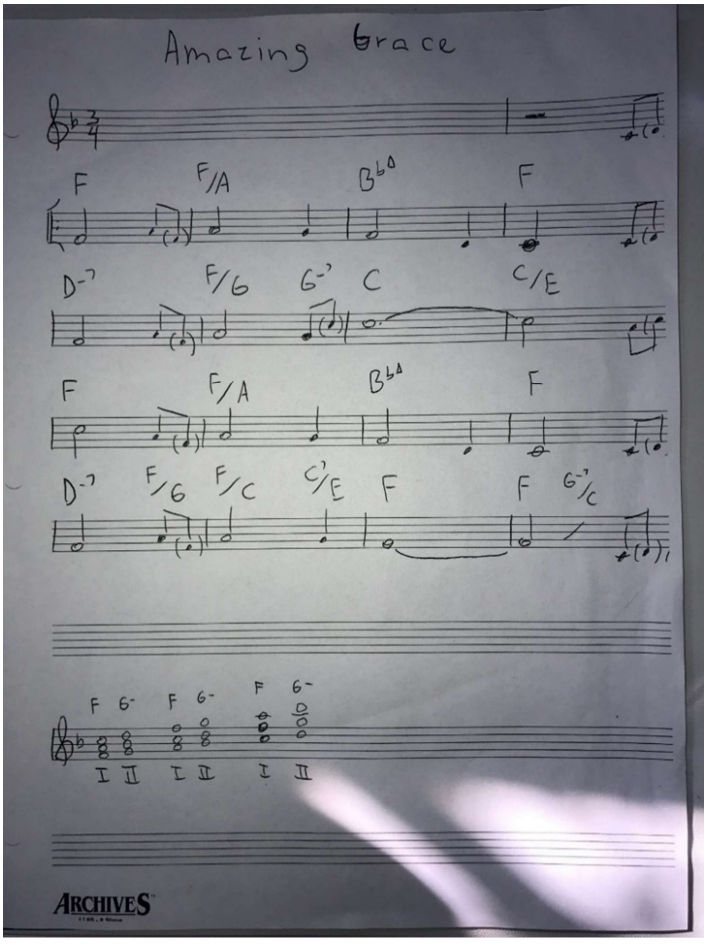


Figure 1. An example of a chart, Amazing Grace.

Finding the Key

I. Look at the chords and use a scale to find the song's key.
    i. Since there is only one flat (Bb major) chord, and F major appears frequently, we can guess that this is written in F major.

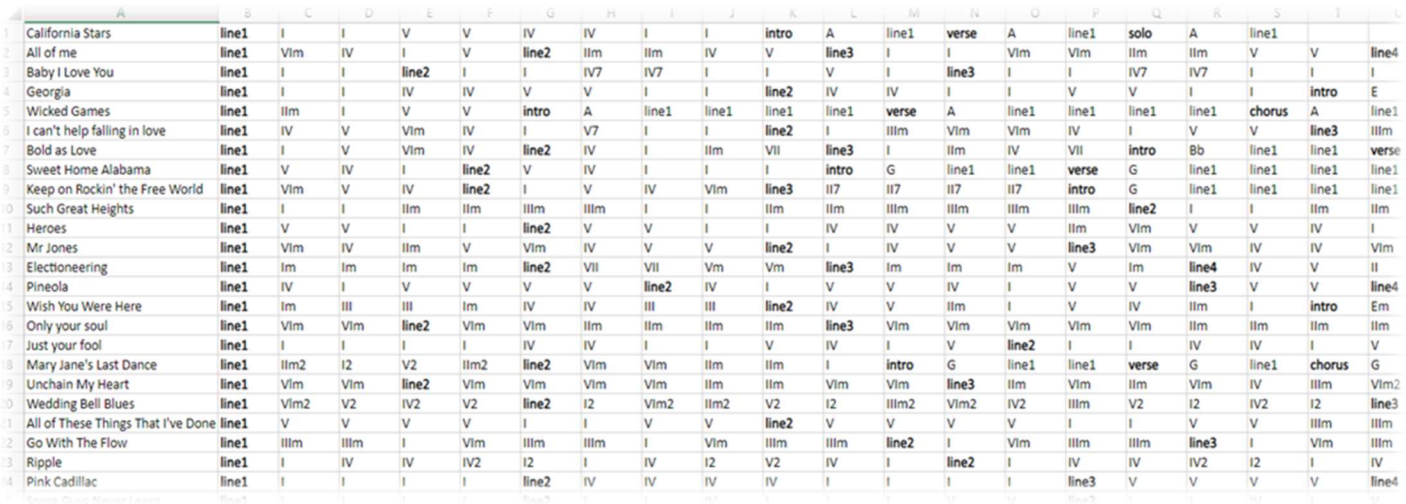
Key	I	ii	iii	IV	V	vi	vii
C Major	C	Dm	Em	F	G	Am	B°
G Major	G	Am	Bm	C	D	Em	F#°
D Major	D	Em	F#m	G	A	Bm	C#°
A Major	A	Bm	C#m	D	E	F#m	G#°
E Major	E	F#m	G#m	A	B	C#m	D#°
B Major	B	C#m	D#m	E	F#	G#m	A#°
F# Major	F#	G#m	A#m	B	C#	D#m	E#°
F Major	F	Gm	Am	Bb	C	Dm	C°
Bb Major	Bb	Cm	Dm	Eb	F	Gm	A°
Eb Major	Eb	Fm	Gm	Ab	Bb	Cm	D°
Ab Major	Ab	Bbm	Cm	Db	Eb	Fm	G°
Db Major	Db	Ebm	Fm	Gb	Ab	Bbm	C°
Gb Major	Gb	Abm	Bbm	Cb	Db	Ebm	F°
Cb Major	Cb	Dbm	Ebm	Fb	Gb	Abm	Bb°

Figure 2. This chart was used as a reference for finding the corresponding Roman numerals for each chord given a key.

Finding the Chord Progression

I. Looking at the same chart, use the key to figure out the numerals used to represent each key.
    i. Since this is in F, the first chord F major is I.
II. We can ignore the special notation on the second chord. Since it is based on the F major chord, it is also I.
III. Bb major in F is IV.
IV. If you are confused between several different keys, try finding the chord progressions for each key and compare the two.
    i. You should use the key that has the more common chord progression.
V. The chord progression for this song is as follows: I-I-IV-I, VIm-I-IIm-V, I-I-IV-I, VIm-I-V-I

Final Version



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
1	California Stars	line1	I	I	V	V	IV	IV	I	I	intro	A	line1	verse	A	line1	solo	A	line1		
2	All of me	line1	VIm	IV	I	V	line2	IIm	IIm	IV	V	line3	I	I	VIm	VIm	IIm	IIm	V	V	line4
3	Baby I Love You	line1	I	I	line2	I	I	IV7	IV7	I	I	V	I	line3	I	I	IV7	IV7	I	I	I
4	Georgia	line1	I	I	IV	IV	V	V	I	I	line2	IV	IV	I	I	V	V	I	I	intro	E
5	Wicked Games	line1	IIm	I	V	V	intro	A	line1	line1	line1	line1	verse	A	line1	line1	line1	line1	chorus	A	line1
6	I can't help falling in love	line1	IV	V	VIm	IV	I	V7	I	I	line2	I	IIIm	VIm	VIm	IV	I	V	V	line3	IIIm
7	Bold as Love	line1	I	V	VIm	IV	line2	IV	I	IIm	VII	line3	I	IIm	IV	VII	intro	Bb	line1	line1	verse
8	Sweet Home Alabama	line1	V	IV	I	line2	V	IV	I	I	I	intro	G	line1	line1	verse	G	line1	line1	line1	line1
9	Keep on Rockin' the Free World	line1	VIm	V	IV	line2	I	V	IV	VIm	line3	II7	II7	II7	II7	intro	G	line1	line1	line1	line1
10	Such Great Heights	line1	I	I	IIm	IIm	IIIm	IIIm	I	I	IIm	IIm	IIIm	IIIm	IIIm	IIIm	line2	I	I	IIm	IIm
11	Heroes	line1	V	V	I	I	line2	V	V	I	I	IV	IV	V	V	IIm	VIm	V	V	IV	I
12	Mr Jones	line1	VIm	IV	IIm	V	VIm	IV	V	V	line2	I	IV	V	V	line3	VIm	VIm	IV	IV	VIm
13	Electioneering	line1	Im	Im	Im	Im	line2	VII	VII	Vm	Vm	line3	Im	Im	Im	V	Im	line4	IV	V	II
14	Pineola	line1	IV	I	V	V	V	V	line2	IV	I	V	V	IV	I	V	V	line3	V	V	line4
15	Wish You Were Here	line1	Im	III	III	Im	IV	IV	III	III	line2	IV	V	IIm	I	V	IV	IIm	I	intro	Em
16	Only your soul	line1	VIm	VIm	line2	VIm	VIm	IIm	IIm	IIm	IIm	line3	VIm	VIm	VIm	VIm	VIm	IIm	IIm	IIm	IIm
17	Just your fool	line1	I	I	I	I	IV	IV	I	I	V	IV	I	V	line2	I	I	IV	IV	I	V
18	Mary Jane's Last Dance	line1	IIm2	I2	V2	IIm2	line2	VIm	VIm	IIm	IIm	I	intro	G	line1	line1	verse	G	line1	chorus	G
19	Unchain My Heart	line1	VIm	VIm	line2	VIm	VIm	VIm	VIm	IIm	IIm	VIm	VIm	line3	IIm	VIm	IIm	VIm	IV	IIIm	VIm2
20	Wedding Bell Blues	line1	VIm2	V2	IV2	V2	line2	I2	VIm2	IIm2	V2	I2	IIIm2	VIm2	IV2	IIIm	V2	I2	IV2	I2	line3
21	All of These Things That I've Done	line1	V	V	V	V	I	I	V	V	line2	V	V	V	V	I	I	V	V	IIIm	IIIm
22	Go With The Flow	line1	IIIm	IIIm	I	VIm	IIIm	IIIm	I	VIm	IIIm	IIIm	line2	I	VIm	IIIm	IIIm	line3	I	VIm	IIIm
23	Ripple	line1	I	IV	IV	IV2	I2	I	IV	I2	V2	IV	I	line2	I	IV	IV	IV2	I2	I	IV
24	Pink Cadillac	line1	I	I	I	I	line2	IV	IV	IV	IV	I	I	I	I	line3	V	V	V	V	line4

Figure 3. The Digital Representation Scheme spreadsheet

This is a spreadsheet file containing song information collected from the charts. This data representation comprises a song model of Roman numeral chord progressions. We represented the songs by translating the chords that were initially written in musical alphabets to Roman numerals. Each chord was then contained in a set of lines. Each line includes a list of chords as well as the number of beats adjacent to the chords if the number of beats is not equal to four. The lines are then contained in a form, along with the song's key. The form then makes up the structure of the song. We chose this strategy because it allows us to easily see the unique lines in each song, and it allows us to easily extract important information that can be useful for the prediction. When using the Digital Representation with the algorithm, the excel spreadsheet is exported to a CSV, comma-separated values, file named Digital Representation.csv and put in the location of the program.

Data Entry

To enter new data into the spreadsheet, follow these steps:

1) Add the name of the song in the first column
2) Add all the unique lines for the song
    a. For each unique line in the song enter “line” with a counter at the end. For example, the first unique line will be called “line1” and the second “line2”.
    b. Following that line name, add each progression in that line in the Roman numeral format with an ‘m’ appended on the end if the chord is a minor.
3) Add the Forms from the song
    a. For each unique From of the song add the Tag (E.g. “verse”)
    b. Then add the Key
    c. Then add the lines that are in that form in order.

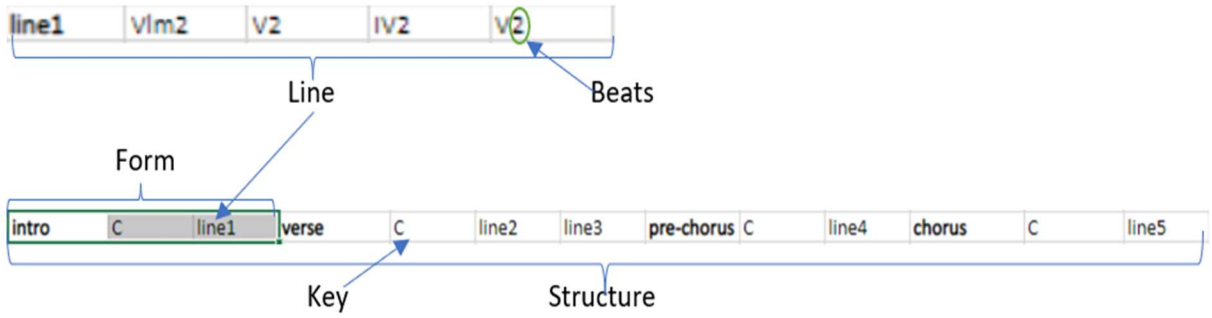


Figure 4. Beats, when less than four, are included after the Roman numeral representing the chord in line definitions. These lines then form the basis of a Form.

Initial Version

Song Name	Key	Chord Progression 1																																				
All of Me	C	VI	IV	I	V	C	VI	IV	I	V	C	VI	IV	I	V	C	VI	IV	I	V	C	VI	IV	I	V	C	II	II	I	V	C	I	I	VI	VI	II	II	V	V	C
All of These Things That I've Done	B	V	V	V	V	I	I	V	V	B	V	V	V	V	I	I	V	V	B	V	V	V	V	I	I	V	V	B	V	V	V	V	I	I	V	V	B	V	V	V
Amazing Grace	F	I	I	IV	I	F	VIm	I	IIm	V	V	F	I	I	IV	I	F	VIm	I	I	V	I	I	IIm															
Baby I Love You	F	I	I	F	I	I	IV	IV	I	I	V	I	F	I	I	IV	IV	I	I	I	IV	IV	V?																
Bold as Love	Cm	III	VII	I	VI	Cm	III	VII	I	VI	Fm	III	VII	I	VI	Cm	III	VII	I	VI	Cm	III	VII	I	VI	Cm	III	VII	I	VI	Fm	VII	I	III	VI	Cm	III	VII	I	VI
California Stars	A	I	I	V	V	IV	IV	I	I																															
Desperado	C	I	IV	I	VIm	II	V	C	I	IV	I	VIm	II	V	C	I	IV	I	IIIm	VIm	IIm	V	I	V	C	VIm	IIIm	IV	I	VIm	IV	I	V	C	VIm	IIIm	IV	I	VIm	IIm
Georgia	E	I	I	IV	IV	V	V	I	I	E	I	I	IV	IV	V	V	I	I	E	IV	IV	I	I	V	V	I	I	E	IV	IV	I	I	V	V	I	I	E	I	I	IV
Go With The Flow	C	IIIm	IIIm	I	VIm	IIIm	IIIm	I	VIm	IIIm	IIIm	C	I	VIm	IIIm	IIIm	C	I	VIm	IIIm	IIIm	C	I	VIm	IIIm	IIIm	C	I	VIm	IIIm	IIIm	IIIm	IIIm							
Havana Daydreamin	A	I	IV	V	IV	A	I	IV	V	IV	V	IV	A	IV	IV	A	I	A	V	V	IV	IV	A	I	IV	V	IV	V	IV	A	I	IV	V	IV	V	IV	A	IV	IV	A
Heroes	G	V	V	I	I	G	V	V	I	I	IV	IV	V	V	IIm	VIm	V	V	IV	I	V	V	G	V	V	I	I	G	V	V	I	I	G	V	V	I	I	G	V	V
Honky Tonk Woman	C	V	V	V	V	C	V	V	I	I	V	C	V	V	I	I	V	V	V	C	V	V	V	V	V	V	C	V	V	I	I	V	C	V	V	I	I	V	V	V
I Can't Help Falling In Love	C	IV	V	VIm	IV	I	V	I	I	C	I	IIIm	VIm	VIm	IV	I	V	V	C	IV	V	VIm	IV	I	V	I	I	C	IIIm	VII	VII	VII	IIIm	VIm	IIm	V	C	I	IIIm	VIm
Jumpin Jack Flash	Ab	IIm	V	I	IIm	V	I	IIm	V	I	IIm	IIm	IIm	IIm	Ab	IIm	IIm	IIm	IIm	IIm	IIm	IIm	IIm	IIm	IIm	IIm	IIm	IV	I	V	IIm	IV	I	V	IIm	IIm	IIm			
Just Your Fool	F	I	I	I	I	IV	IV	I	I	V	IV	I	V	F	I	I	IV	IV	I	V	I	V	F	IV	`I	IV	I	IV	I	V										
Keep On Rockin In The Free World	G	VIm	V	IV	G	VIm	V	IV	VIm	V	IV	G	I	V	IV	VIm	G	IIm																							
Kiss	A	V	A	I	I	I	I	A	I	I	I	I	I	I	I	I	IV	IV	IV	IV	I	I	I	I	A	V	V	IV	IV	V	V	IV	I	A	I	I	I	I	I	I
Left Hand Free	Eb	IIm	I	V	IIm	Eb	IV	V	Eb	IIm	I	V	IIm	IV	V	Eb	II	VII																						
Little Red Fish and the Devil's Slide	G	V	V	V	V	G	I	V	V	I	I	V	V	IIm	I	V	V	G	I	V	V	I	I	V	V	IIm	IIm	V	V	G	I	V	V	I	I	V	V	IIm	IIm	V
Losing Keys	G	I	IV	I	IV	I	IV	IIm	V	IIm	V	I	I	V	VIm	IV	I	V	VIm	IV	IIm	V	IIm	V	IV	IV	V	I	I	V	VIm	IV								
Louise	D	V	V	VIm	VIm	I	IIm	V	V	I	IIm	V	V	I	IIm	V	V	I	IIm	IIIm	IIIm	VIm	VIm	IIm	IIm	V	V	VIm	VIm	I	IIIm	V	V								
Mary Jane's Last Dance	G	IIm	I	V	IIm	G	VIm	VIm	IIm	IIm	I	G	VIm	I	IV	VIm																								
Molly	Db	I	VIm	IV	IV	I	VIm	IV	IV	Db	IIm	IIm	IIIm	IIIm	IV	IV	IIIm	IIIm	IIm	IIm	IIIm	IIIm	IV	IV	V	V	Db	I	I	IV	IV	Db	I	VIm	IV	IV	I	VIm	IV	IV
More of You	E	I	I	I	I	E	I	I	I	I	I	I	I	IV	IV	E	I	I	V	V	V	V	I	I	V	V	I	I	IV	IV	I	I	V	V	I	I	V	V	I	I
Mr Jones	C	VIm	IV	IIm	V	VIm	IV	V	V	C	I	IV	V	V	C	VIm	VIm	IV	IV	VIm	VIm	V	V	C	V															
Only Your Soul	C	VIm	VIm	VIm	C	VIm	VIm	IIm	IIm	IIm	IIm	VIm	VIm	VIm	VIm	VIm	IIm	IIm	IIm	IIm	C	I	I	I	C	I	V	V	V	V	VIm	VIm	VIm							
Pineola	C	IV	I	V	V	V	V	C	IV	I	V	V	IV	I	V	V	C	IV	I	V	V	IV	I	V	V	C	IV	I	V	V	IV	I	V	V	C	IV	I	V	V	IV
Pink Cadillac	A	I	I	I	I	A	I	I	I	I	I	I	I	I	IV	IV	IV	IV	I	I	I	I	V	V	V	V	A	I	I	I	I	IV	IV	I	I	V	V	I	I	I
Purple Rain	Bb	I	VIm	V	IV	Bb	I	VIm	V	IV	I	VIm	V	V	I	Bb	IV	IV	I	VIm	V	V	V	I																
Ripple	D	I	IV	IV	IV	I	I	IV	I	V	IV	I	D	IIm	V	I	IV	IIm	V																						
Seven Bridges Road	D	I	IIm	IV	I	D	IIm	I																																
Shine a Light	D	I	I	VIm	V	IV	I	V	IV	I	D	I	I	VIm	V	IV	I	V	IV	I	D	I	V	IV	I	V	D	I	V	IV	I	V	D	I	I	VIm	V	IV	I	V
Simple Things	C	I	I	I	I	C	I	IV	VIm	VII	I	IV	VIm	V	C	I	IV	VIm	VII	IIm	V	V	C	VIm	IIIm	IV	I	IV	I	C	VIm	IIIm	IV	I	VII	IV	C	I	I	I
Some Guys Never Learn	A	I	IV	I	V	A	I	IV	I	V	IV	I	A	I	IV	I	V	A	I	IV	I	V	IV	I	A	I	IV	I	V	A	I	IV	I	V	IV	I	A	VIm	V	VIm
Starlight	E	V	V	V	V	E	V	VIm	IIIm	I	E	V	VIm	IIIm	I	E	V	VIm	IIIm	I	E	V	VIm	IIIm	I	E	VIm	IIm	VII	IIIm	IV	I	IV	VII	E	IIIm	VII	I	V	V

Figure 5. The first version of the Digital Representation Scheme.

This is an initial version of our data representation file. This version also contains the song information that is extracted from the chart but unlike our final model, we did not include the number of beats. Adding the beats is important when playing the music which led us to slightly adjusting our representation. We also did not break down the songs into forms which would have made it harder to extract the information needed for the prediction. This is something we kept because there was still valuable information here about keys and the chord progressions for all of these songs that did not necessarily end up in our updated digital corpus.

Objects in Python

One way to represent the songs is to break each song into an ordered list of progressions which each contain a collection of lines. Each line contains a list of chords which have an integer after the Roman numeral if the number of beats the chord is played is not equal to four. Each progression would have at least one line and be contained in a form which would tag the progression as a specific section of the song. Chord progressions inside the same section are stored inside of the same form. The forms are then stored inside a list as part of the Song object. This is an attractive solution because it is very easy to see the unique lines in each song as well as a higher-level view of how each song is a collection of forms like intro -> verse -> pre-chorus -> chorus. This solution will store all the potentially relevant information while also enabling us to easily extract the unique progressions or all the progressions for each song.

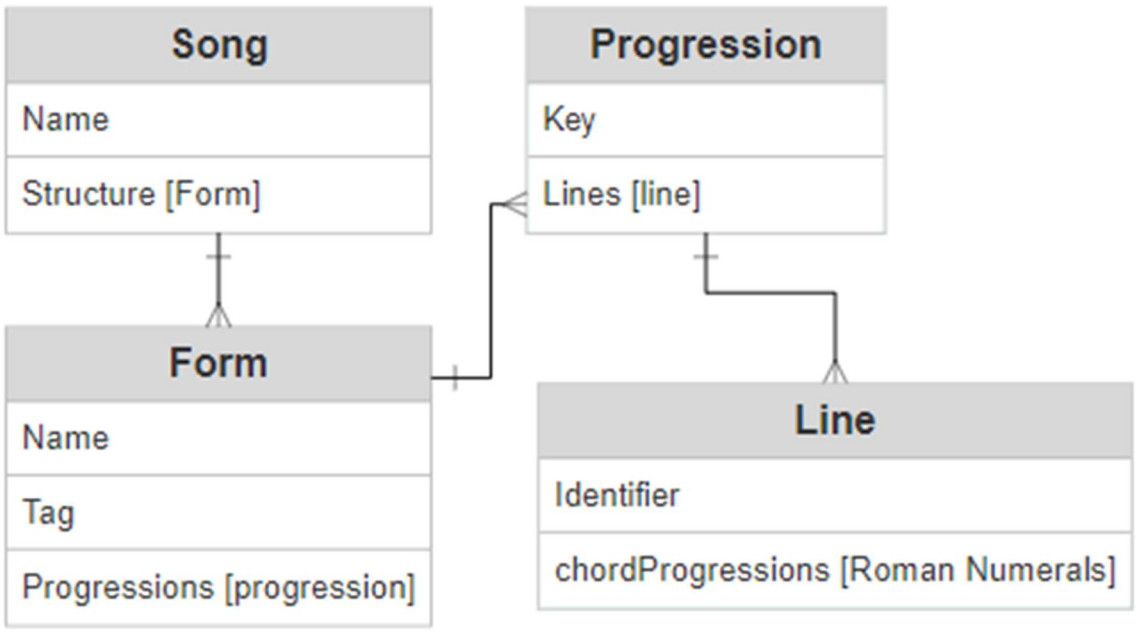


Figure 6. A diagram describing the relationships between each Song object and the data contained.

Here is a snippet of code showing this implemented in Python:

```
class Song:
    def __init__(self, n, f):
        self.name = n
        self.forms = f
        pass


class Progression:
    def __init__(self, k, ln):
        self.key = k
        self.line = ln
        pass


class Line:
    def __init__(self, i, c):
        self.identifier = i
        self.chords = c
        pass


class Form:
    def __init__(self, t, p):
        self.tag = t
        self.chordProgressions = [p]
```

Figure 7. The Python code defines each Song object.

For a more specific example, take the song, “All of Me.” The image below shows how each Song object is created from a section of the song.

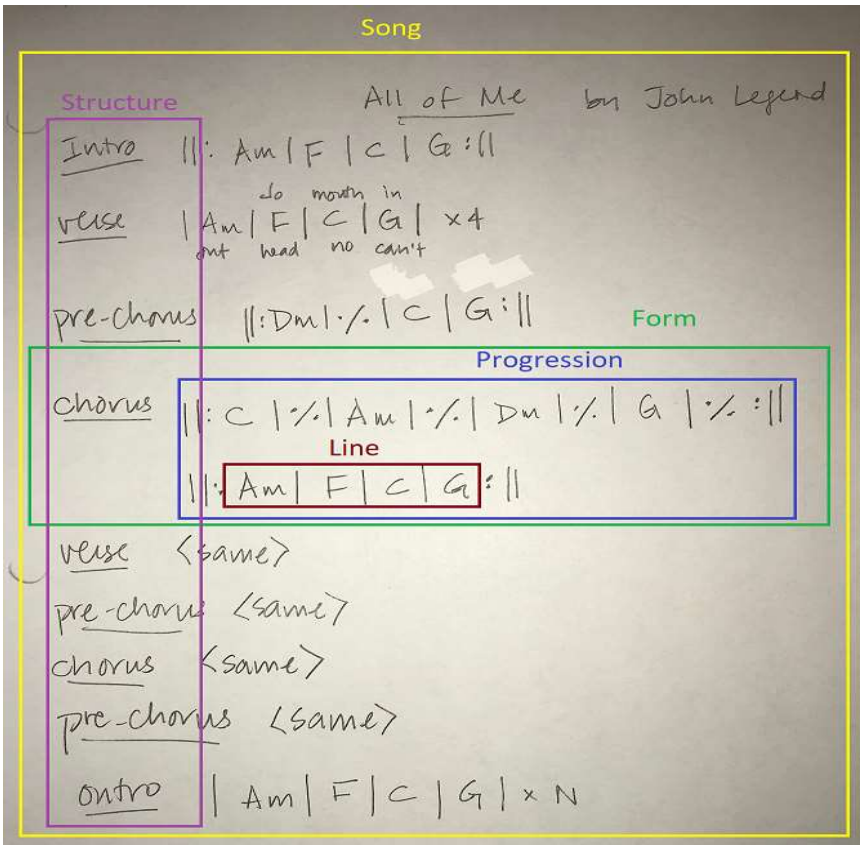


Figure 8. A song chart with colored rectangles highlighting the corresponding object in Python.

Applying this in Python using our Song object definition results in data that looks like the figure below.

```
# Example for song "All of me":

line1 = Line("Line1", ['Vim', 'IV', 'I', 'V'])
line2 = Line("Line2", ['IIm', 'IIm', 'IV', 'V'])
line3 = Line("Line3", ['I', 'I', 'VIm', 'VIm', 'IIm', 'IIm', 'V', 'V'])
line4 = Line("Line4", ['VIm', 'IV', 'I', 'V'])

prog1 = Progression('C', [line1, line1])
prog2 = Progression('C', [line1, line1, line1, line1])
prog3 = Progression('C', [line2, line2])
prog4 = Progression('C', [line3, line3, line4, line4])
prog5 = Progression('C', [line1])
intro = Form(intro, prog1)
verse = Form(verse, prog2)
pre_chorus = Form(pre_chorus, prog3)
chorus = Form(chorus, prog4)
outro = Form(outro, prog5)

song = Song("All of me", [intro, verse, pre_chorus, chorus, outro])
```

Figure 9. An example of how each component of the song, "All of Me," is stored into Python objects.

Object Programming

In order to use the data contained in our spreadsheet, we needed a program to read the data and use it to create each Song object and the corresponding components (Digital Representation Scheme Documentation). The file Object.py contains the program which relies on a helper function, Loader.py, to read the 'Digital Representation Scheme.csv' spreadsheet and turn it into a list of rows, with each row itself a list, that is used as input.

The program iterates over the list from Loader.py extracting the information from each element, creating a sub list, and creating each object before moving on to the next element. Several pieces of information are extracted from each list: the song name, a list of the chords in each line, the lines in each progression, and the names of each form in the song. The extraction is accomplished inside of the main while loop where each conditional block marks the start or end of a line or chord progression. The lines and progressions are stored as a list inside a nested list since songs typically have more than one line and progression.

object.py | Digital Representation.csv

```
import loader
import Progressions


def main():
    '''
    Returns list of Song Objects from nested list output by loader.py

        Parameters:
            None

        Returns:
            output (list): List of Song Objects (See Documentation for details)
    '''
    songList = loader.main()
    output = [] #list of Song objects
    for i in songList: #iterates through file by row
        lines = []
        progs = []
        forms = []
        name = i[0]
        j = 0
        while j < (len(i)-1): #iterates through each cell in row
            line = []
            if "line" in i[j] and ("I" in i[j+1] or "V" in i[j+1]): #Marking start of line
                x = j
                j += 1
                while j < i.index("intro"):
                    if "line" not in i[j]:
                        j += 1
                    else: #marking end of line, start of next line
                        lines.append(i[x:j])
                        x = j
                        j += 1
                lines.append(i[x:j])
            elif "line" not in i[j] and len(i[j+1]) <= 3: #Check for start of progression
                z = j
                j += 1
                success = False
                while j < (len(i)-2):
                    if "line" in i[j] and "line" not in i[j+1]: #Check for end of progression
                        progs.append(i[z:j+1])
                        success = True
                        break
                    else:
                        j += 1
                if not success: #Grabs last progression
                    progs.append(i[z:])
            else:
                j += 1
```

Figure 10. The first section of object.py, which is responsible for extracting data from the spreadsheet and storing it inside lists.

The second section is responsible for creating the Python objects using the classes as defined in Progressions.py. Each list is iterated through using a for loop. The line objects are created first before replacing the strings representing each line object in the progressions list. The progressions are then created using the progressions list and then added to the forms list. A progression with a name that already appears in the forms list is added to the same form as an additional progression. Finally, the Song is created using the forms list and the song name and then added to the output list of songs.

```
        for k in range(len(lines)):
            lines[k] = Progressions.Line(lines[k][0], lines[k][1:])
        for l in range(len(progs)):
            for o in range(len(progs[l])):
                if "line" in progs[l][o]: #Replaces line strings with corresponding Line objects
                    progs[l][o] = lines[int(progs[l][o][-1])-1]
            progs[l][1] = Progressions.Progression(progs[l][1], progs[l][2:])
            progs[l] = progs[l][:2]
            if forms:
                for n in forms:
                    if (progs[l][0] == n.tag):
                        for q in n.chordProgressions:
                            if (q != progs[l][-1]): #Repeat progressions under the same tag are not allowed
                                n.chordProgressions.append(progs[l][-1])
                    elif n == forms[-1]:
                        forms.append(Progressions.Form(progs[l][0], progs[l][-1]))
            else:
                forms.append(Progressions.Form(progs[l][0], progs[l][-1]))
        output.append(Progressions.Song(name, forms))
    return output
```

Figure 11. The second section of object.py creates the Python objects from the lists created in the first section.

Algorithm Documentation

We have implemented our algorithm using the digital representation scheme inside of a GitHub repository. After researching different machine learning solutions for predicting the next item in a sequence, we ended up writing an algorithm that uses Bayesian probability to make predictions. Our algorithm is based on a Markov Chain model with the Bayesian probability using n-grams from the data.

We used Bayesian probability of a trigram → P(C|AB)

Bayesian probability of a bigram → P(B|A)

Prior probability → P(A)

Example with string = 'ABCDEFG'

Conditional probability:

P(G|ABCDEF) → Probability of G given ABCDEF = Probability of the entire string

Probability of the entire string:

1. P(ABCDEFG)

   ↓

2. P(G|ABCDEF) P(F|ABCDE) P(E|ABCD) P(D|ABC) P(C|AB) P(B|A) P(A)

   ↓

3. P(G|EF) P(F|DE) P(E|CD) P(C|AB) P(B|A) P(A) P(A|<st>)

   *<st> → probability of a string = 1

Method 3 has a close enough probability to method 2 with much simpler expression.

Example with two different chord progressions:

(1) I – IV – I – V (2) I – IV – I

Table 1. A table describing the relationships between each probability and the chords.

Probability	Chords	N - gram	Count for (1)	Count for (1) & (2)
$P(I)$	I	Unigram	2	4
$P(IV))$	IV		1	2
$P(V)$	V		1	1
$P(IV \mid I)$	I – IV	Diagram	1	2
$P(I \mid IV)$	IV - I		1	2
$P(V \mid I)$	I - V		1	1
$P(I \mid IV\ I)$	I – IV - I	Trigram	1	2
$P(V \mid IV\ I)$	IV – I - V		1	0

We can see the general pattern of the count decreasing as the chord progression branches out.

When we added a second chord progression (2), we ended up with brand new trigram and increased frequency count.

↳ We are looking at the frequency of one, probability of given previous one.

Things to consider:

1. What if there is no value for P(E|CD)?

   If there is no value for P(E|CD), then the whole probability would be multiplied by 0.

   P(E|CD) == 0 → P(G|EF) P(F|DE) **P(E|CD) = 0** P(C|AB) P(B|A) P(A) P(A|<st>)

   ↳ We could take a step back, find the probability of a bigram rather than a trigram, and use that value instead.

   P(E|CD) → P(E|D)

   What if there's no value for P(E|D) either?

   ↳ We could take a further step back, find the probability of a unigram, and use that value instead.

   P(E|D) → P(E)

2. We could predict less predictable things because we count it to be as far from typical as possible by looking at the possibility that it is possible, but less likely to happen.
3. We could think about what the training data gives us. *How many example trigrams can we actually watch?*
4. We might see something that is possible based on the theory that we do not have in our data.

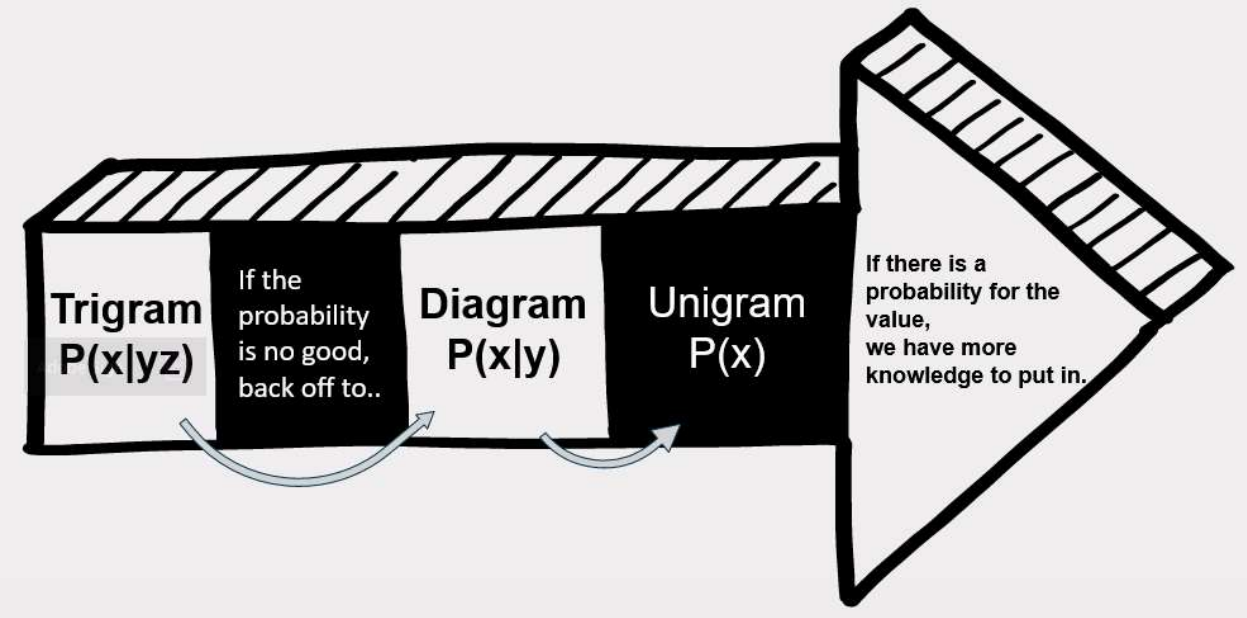


Figure 12. Describes the process of using less "accurate" probabilities given a lack of information.

We have touched a basic level of language models using statistical language modeling with n-gram.

Recommender systems, finite state model, and language models were suggested the most by industry experts for further future development of the algorithm. Since there is not an ample dataset, we decided to stick with much simpler model (n = 3).

Algorithm Analysis

```python
def possibleProgressions(possibleChords):
    """Calls getLines() and counts how many times each chord occurs to find P(A)..."""

    # List of each unique line from the dataset
    lines = getLines.main()
    # list of progressions in a line
    progCount = [0] * len(possibleChords)

    for l in lines:
        for c in range(len(l.chords)):
            # Add progressions that are present in a line
            progCount[possibleChords.index(l.chords[c])] += 1

    return progCount
```

Figure 13. possibleProgressions function inside of algorithm1.py

Big O Notation for possibleProgressions: O(n) given n is the number of chords from all the lines

```python
def possibleProgressionsLast1(possibleChords, inProg):
    """..."""

    # List of each unique line from the dataset
    lines = getLines.main()

    # list of possible following progressions
    progCount = [0] * len(possibleChords)

    for l in lines:
        for c in range(len(l.chords)):
            # If the progression from inProg is there and not at the end of a line,
            # add the following element to possibleFollow
            if l.chords[c] == inProg and c != len(l.chords) - 1:
                progCount[possibleChords.index(l.chords[c + 1])] += 1

    return progCount
```

Figure 14. possibleProgressionsLast1 function inside of algorithm1.py

Big O Notation for possibleProgressionsLast1: O(n) given n is the number of chords from all the lines

```
def possibleProgressionsLast2(possibleChords, inProg1, inProg2):
    """..."""

    # List of each unique line from the dataset
    lines = getLines.main()

    progCount = [0] * len(possibleChords)

    for l in lines:
        for c in range(len(l.chords)):
            # If the progression from inProg is there and not at the end of a line,
            # add the following element to possibleFollow
            if l.chords[c] == inProg1 and c < len(l.chords) - 2 and l.chords[c + 1] == inProg2:
                progCount[possibleChords.index(l.chords[c + 2])] += 1

    return progCount
```

Figure 15. possibleProgressionsLast2 inside of algorithm1.py

Big O Notation for possibleProgressionsLast2: O(n) given n is the number of chords from all the lines

```
def baysianModel(possibleChords):
    """..."""

    alg0Out = normalize.main(algorithm1.possibleProgressions(possibleChords))
    alg1Out = []
    alg2Out = []

    for chord in possibleChords:
        alg1Out.append(normalize.main(algorithm1.possibleProgressionsLast1(possibleChords, chord)))

    for chord1 in possibleChords:
        for chord2 in possibleChords:
            alg2Out.append(normalize.main(algorithm1.possibleProgressionsLast2(possibleChords, chord1, chord2)))

    return [alg0Out, alg1Out, alg2Out]
```

Figure 16. baysianModel inside of predictor.py

Big O Notation of baysianModel: $O(n^3)$

Process Flow Chart

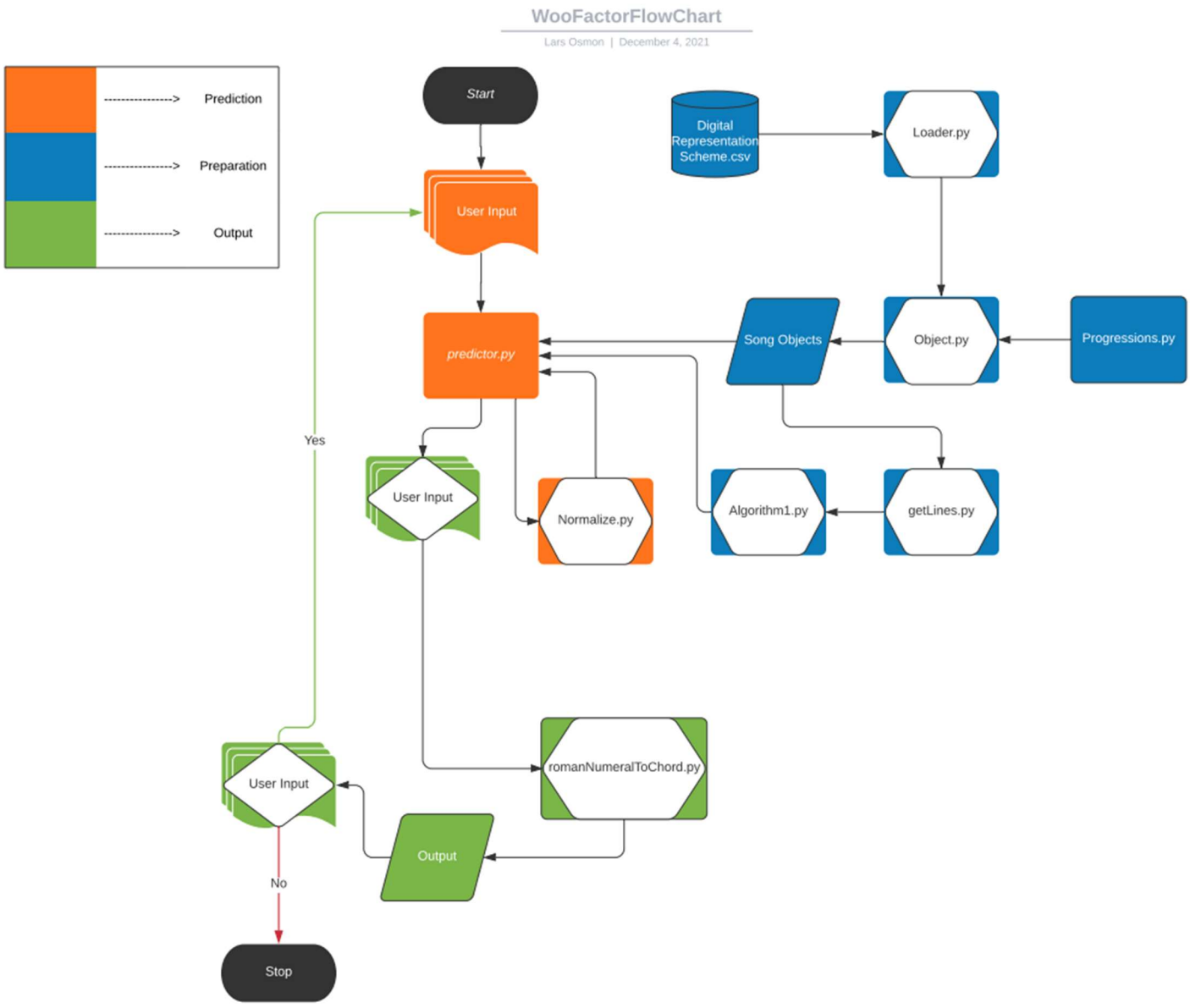


Figure 13. A flowchart depicting algorithm execution and training. Hexagons represent data preparation and triangles represent decisions. Online Chart

User Guide

Installation Steps

1. Clone or download the repository

   Navigate to the Code tab in the GitHub repository and either download the ZIP or copy the URL for cloning. If cloning, navigate to the desired directory and execute “git clone [url]” to grab the latest version otherwise unzip the downloaded file.

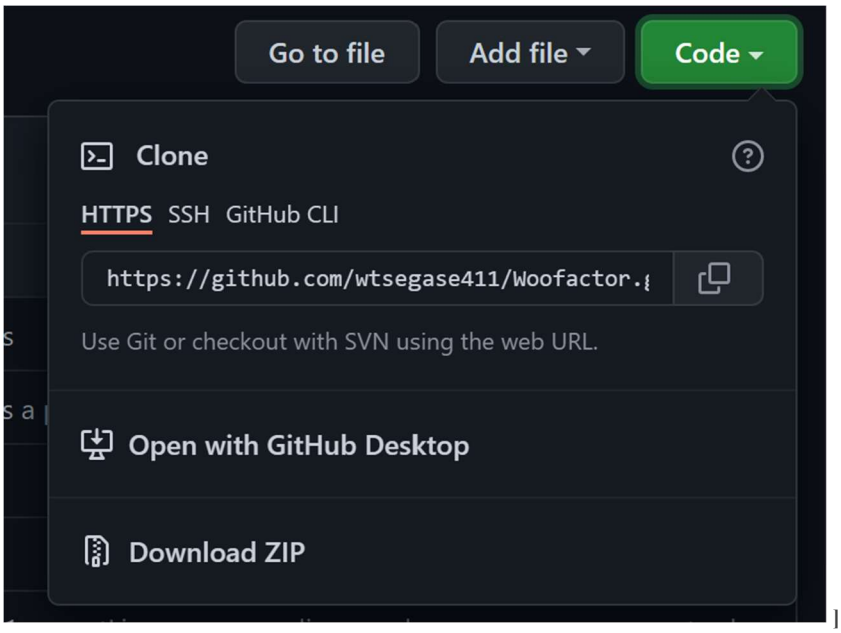


Figure 14. The code tab allows the user to clone the GitHub repository with the URL or download the repository as a ZIP file.

Installing Packages

The following packages are required: matplotlib. Either install using the command line with the command “pip install [package name]” or install via an IDE like PyCharm.

Running the Predictor

The main algorithm that the user will interface with is predictor.py either inside of the directory or through an IDE run predictor.py.

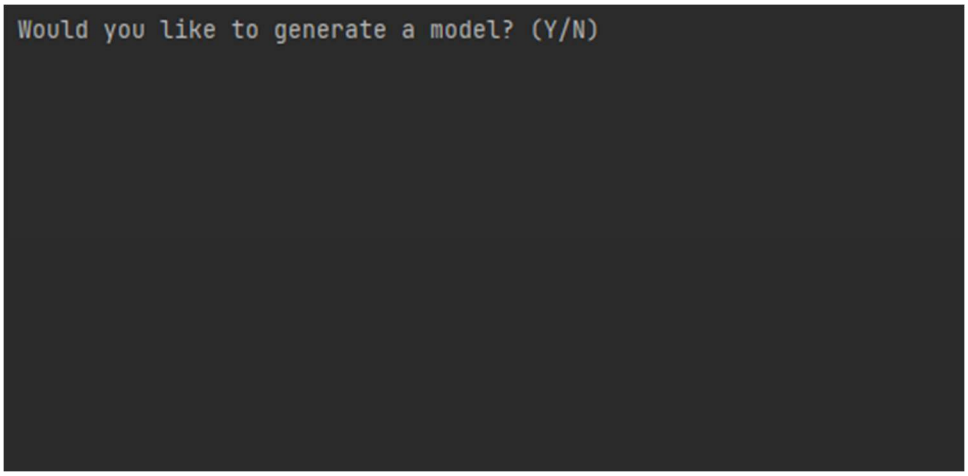


Figure 17. The first prompt when running the algorithm in a terminal. It determines whether the algorithm is trained or not.

The user is prompted to generate a model. If this is the first time predictor.py is run, then the user will need to input "Y" to create a model based on the existing data. Any time data is adjusted, a new model should be generated, however, given an invalid prompt, the value will default to N.

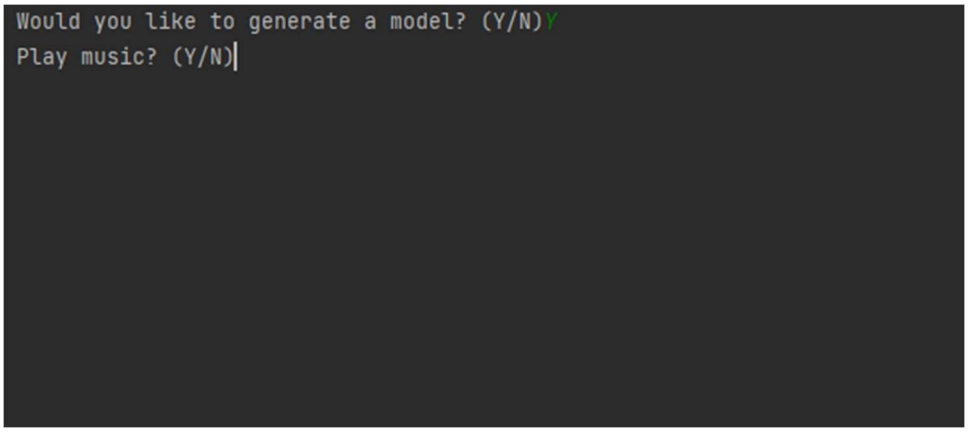


Figure 18. The second prompt determines if the output is played using the synthesizer

The next prompt determines whether the output from the algorithm will be played aloud by the synthesizer. The default given an invalid prompt is N.

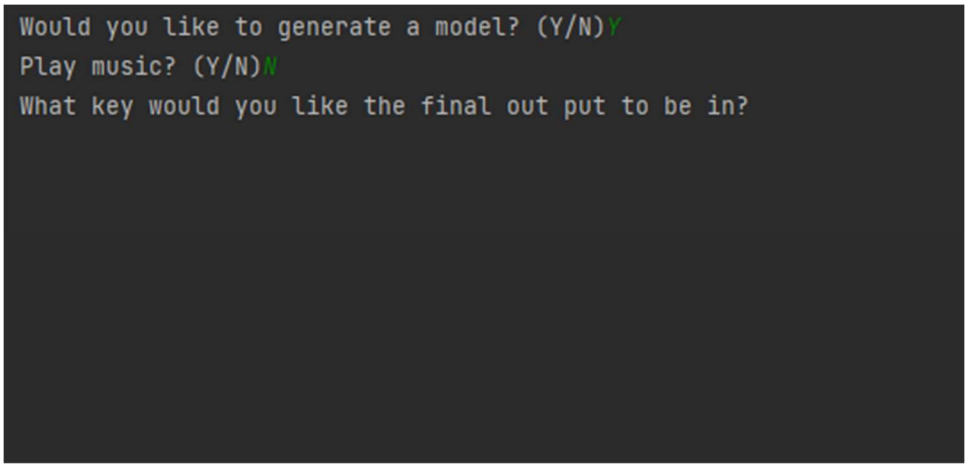


Figure 19. The third prompt allows the user to decide the key that the output uses to find the chord names.

The user is then given the choice of the key of the output. Any major or minor key is valid, and the user will be prompted again if the input is invalid. A major key (e.g., A, B, C, D, E, F, G) can be changed into a minor key by including an ‘m’ after the key.

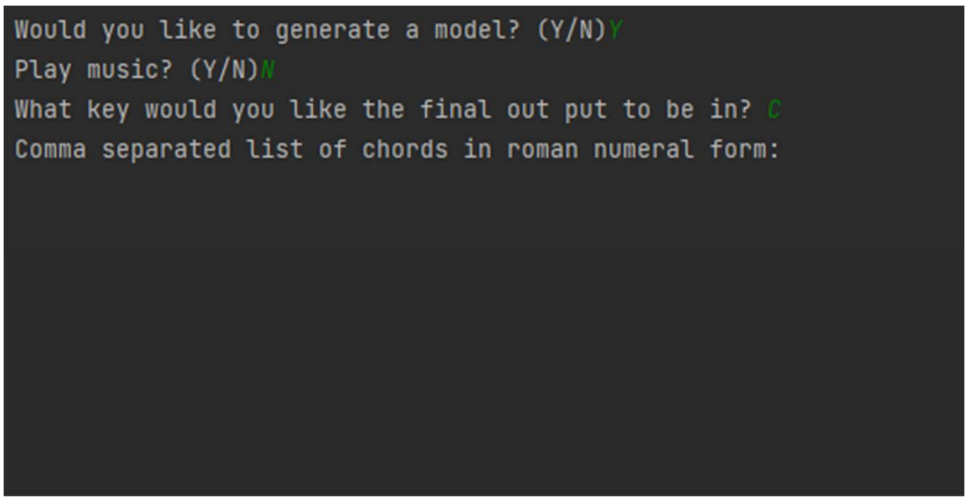


Figure 20. The fourth prompt lets the user input several chords to use for predicting future chords.

The next prompt is for the initial data provided to the algorithm. The user must input a comma-separated list of chords in the form of Roman numerals. This input will be used to predict the next chord progression.

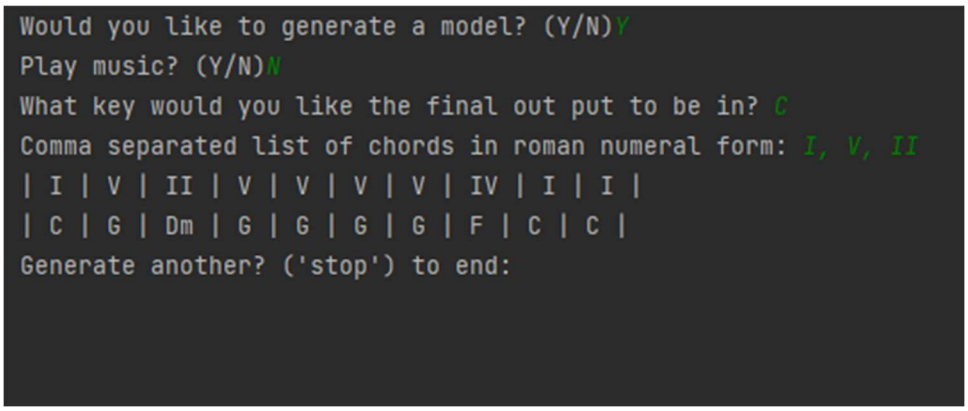


Figure 21. The final prompt allows the user to run the algorithm prediction loop again or terminate the program.

The final prompt allows the user to generate another progression from new input, otherwise the program will terminate when the user inputs “stop.”