

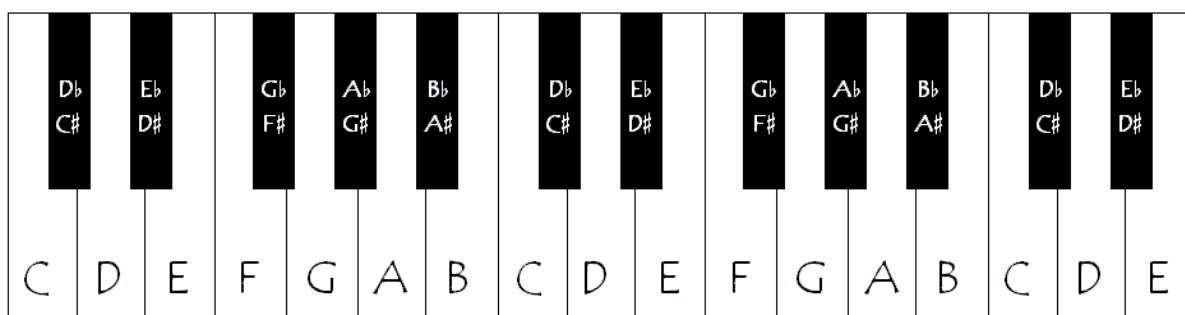


## CHAPTER-1

### INTRODUCTION

Music is a domain which can be interpreted differently by practitioners of different professions. To an orthodox musician, music is an artistic sequence of swaras or notes grouped together as a raga to form melodious tunes. However, to a technically inclined engineer, music is nothing but sound. Music can be treated as a set of frequencies generated at certain amplitudes. These frequencies can be generated in multiple ranges, and music has the inherent property that doubling a frequency produces the same note albeit in a higher octave. It is this frequency range that determines the characteristics of a musical instrument. Even though the same sequence of notes can be played on a flute as well as a bass guitar, they will sound distinctly different due to this property of varying frequency ranges. In the context of this project, the primary musical instrument of concern is the keyboard.

A keyboard is an electronic version of a piano, and is programmed to produce certain frequencies when each key is pressed. It was designed as a cheaper and more practical alternative to a piano, which has a set of strings precisely tightened in order to generate required frequencies when hit by an internal hammer which is triggered upon a key press. A keyboard and piano do not differ in terms of the interface they provide to a user. They both have evenly spaced black and white keys representing musical notes as shown in the figure below.



**Fig. 1.1 Keyboard**

There are twelve swaras in music, and these are given different names across the world.

Do	Shadja (Sa)	C
Do Diese	Komal Rishabha (Komal Ri)	C#
Re	Shuddha Rishabha (Shuddha Ri)	D
Re Diese	Komal Gandhaara (Komal Ga)	D#
Mi	Shuddha Gandhaara (Shuddha Ga)	E
Fa	Shuddha Madhyam (Shuddha Ma)	F
Fa Diese	Teevra Madhyam (Teevra Ma)	F#
Sol	Panchama (Pa)	G
Sol Diese	Komal Dhaivata (Komal Dha)	G#
La	Shuddha Dhaivata (Shuddha Dha)	A
La Diese	Komal Nishada (Komal Ni)	A#
Si	Shuddha Nishada (Shuddha Ni)	B

**Table 1.1 Twelve Swaras in Music**

A keyboard sounds the best when played by a professional who makes use of both his hands. Typically, the right hand plays a melody while the left hand accompanies by playing chords or harmonies. The difference between a melody and a harmony is that a melody drives the tune of the song, and defines the characteristics of the song. A harmony on the other hand, is a side-kick to the melody and complements the tune by providing bass tones which induce the feeling of “depth” in the music. Since the melody and harmony aren’t necessarily the same actions i.e key presses on a piano/keyboard, the art of playing a piano/keyboard takes very long to master due to the need for synchronization of both hands. An amateur most

certainly cannot grasp the intricacies of playing with both hands, and usually resorts to simple melodies played with just the right hand. Since the novice is not skilled enough to use his left hand effectively, the music he produces ends up sounding stale and uninteresting. This inevitably leads to demotivation of the learner, and results in quitting of music lessons. Having all these ideas in mind, we proceed to tackle this commonly observed problem in the field of music, specifically keyboard.

## CHAPTER-2

### PROBLEM DEFINITION

The aim of this project is to generate chords and arpeggios to accompany the notes the aforementioned amateur plays on the right side of the keyboard, in real-time. This would enable the user to feel more “performance-ready” as he/she would get a feel of sounding like a professional when the left hand notes are filled in by the model. Apart from motivation to continue playing, this project provides an opportunity for the amateur to view the chords that were generated in order to convert the process into a learning experience.

In order to achieve this goal of real-time generation, the focus of the project is on learning the concept of chord progressions, which is backed by music theory, for appropriate scales and mapping them to the notes the end user plays.

---

## CHAPTER-3

### DEFINITIONS, ACRONYMS AND ABBREVIATIONS

- Sheet Notation : A representation of music to be played, in terms of symbols rather than letters (like Sa Re Ga Ma or C D E F)
- Treble Clef : This refers to anything played on the right side i.e high frequency notes.
- Bass (pronounced base) Clef : This refers to anything played on the left side i.e low frequency notes.
- Chord : A chord is a set of three or more notes played together to produce a harmony. It can't be just any three notes; there are specific patterns/rules to form a chord. There are different kinds of chords : Major, Minor, Sharp, Augmented, Diminished, Seventh, etc
- Broken Chord or Arpeggio : As the name suggests, the notes that were played together in a chord are played one after the other to form a broken chord. Chords (broken or otherwise) are typically played with the left hand.
- Beat : Each snap of your finger is a beat. For example, in the first line of Twinkle Twinkle, one would snap their fingers at the beginning of each word in "Twinkle Twinkle Little Star". For the second line, a beat starts at each of "How I" - "Wonder" - "What you" - "Are"
- Tempo : Tempo is a number defined as BPM or Beats Per Minute. The faster a song is, the faster you snap your fingers, the more snaps you make in a minute, the higher the tempo.
- Bar : A bar is a unit of time for a song. It is not a general time unit i.e a bar is not a fixed length of time like 2 seconds. A bar can be X seconds long for a song with tempo Y.

## CHAPTER-4

### LITERATURE SURVEY

The cyclic tempogram [1] paper is concerned with generating a tempogram, defined as a ‘time-tempo representation for a given time-dependent signal’. Here a tempogram is considered as a tempo analogue of a spectrogram of frequencies, where at each discrete time interval a given tempo will be dominant and other tempos will be subservient. However, due to the presence of several pulse levels such as the measure, tactus (the principle accent of a rhythm unit) and tatum levels (lowest regular pulse train that a listener intuitively infers) that contribute to the human perception of tempo, ambiguities arise. These pulse levels can be directly compared to harmonics in the context of pitch, such that for a given tempo, one has both several multiples ( $T, 2T, 3T, \dots$ ) and divisions ( $T, T/2, T/3, \dots$ ) accompanying it. In order to deal with this ambiguity cyclic tempograms are created by creating tempo equivalence classes for such tempos.

In order to detect local tempos, a ‘novelty curve’ is used, that is generated every time a note-onset event occurs. Then using autocorrelation, comb-filter, and Fourier methods, periodic patterns are extracted in order to create a tempogram.

The chord generation [2] paper applies machine learning techniques to follow measures of melody and generate pleasant music pieces. As a dataset it uses 43 lead sheets, of which almost all have only one chord corresponding to one measure, and only seven scale tone chord classes are used. Each song is shifted in the C chord, while maintaining one chord per measure. Features used include the note value, the notes on the beat in the measure, and the longest note in the measure. Models used include Logistic Regression, Naive Bayes, SVM with linear, Polynomial and Radial Kernels, Random Forests and Boosting. Random Forests and Boosting are the most solid models with the best results, followed by SVM with Radial Kernel.

The hybrid model for prediction [3] proposes a combination of two techniques - machine learning based neural network and a rule-based sequence tracker - are used for the purpose of chord prediction in real time. The rule-based sequence tracker detects recurrent chord sequences while the song is being played and uses this information for its purpose. The paper distinguishes between a priori and in-the-flight knowledge in order to predict the appropriate chord. No universal rules exist for chord chaining, and depend on various factors and context such as the song, the composer, and the style it is played in. However some chord sequences are recurrent (II-V-I, II-V) and this information is utilized for the task. The hybrid model proposed works as a competitive system between the neural network predictor and rule-based sequence tracker. The sequence tracker monitors the incoming chords and when it detects a sequence, outputs its corresponding prediction. When it is not sure whether it has detected a pattern, it transfers control to the neural network which predicts a chord based on the data learnt. The dataset used consists of jazz songs and testing is done by predicting the chord present in the original unheard song. The results of the hybrid model gives an error rate of 9 percent on the dataset used.

The neural net approach [4] deals with a neural network modelling relations between three parameters - harmony, melody, and meter. The model learns sequences of harmonized melodies. The corpus contains sixty eight stylized popular western diatonic melodies. The harmonized melodies share a single meter (4/4), a single mode (major), identical lengths (16 measures), identical key (C major) and common structures. The net learns relations between important notes of the melody and their harmonies and is able to produce harmonies for new melodies in real-time, that is, without knowledge of the melodic continuation. The neural network model is fed with a musical score and it learns the sequence of chords as a function of the melody's notes and the metric index.

The context dependent [5] paper aims to replicate the human ability to tap beats to music in order to estimate tempo by performing beat tracking. The approach consists of two states, the



General State and the Context-Dependent State. The onset detections are considered as mid-level representations of the music from which beats are estimated. Given an input onset detection function frame, the role of the General State is to infer the time between successive beats, the beat period, and the offset between the start of the frame and the locations of the beats, the beat alignment, without any prior knowledge of the input. It operates in a memory-less fashion, extracting these features through induction. This can cause errors in cases of switching of metrical levels and switching between the on and off beat. To deal with these limitations, the Context-Dependent State is used. It operates in a similar way to the General State in that it separately extracts the beat period and the beat alignment. Instead, it is based on the simple assumption that, if beats are arriving at regular beat period intervals, then the focus is no longer on inducing beat locations but in maintaining continuity.

Waoon [6] is an open source tool that achieves the wav to notes conversion, by taking a wav file as input and providing the MIDI file as output. MIDI can be understood like the digital alphabet for music. This format encodes a series of messages that tell an electronic device how to generate certain sounds. Thus, since no actual sounds are stored, MIDI files have a very small size.

Although waoon tries to convert the WAV audio file to the MIDI format, it still has some amount of inaccuracies to it, because when the signals are read from the WAV audio file, a single note also consists of its overtones, which will be added to the MIDI, even though it is not played in a pure piano music piece. Apart from this, in musical pieces which involve more instruments, the problem is that such a software has to identify individual musical notes from the audio stream to convert them into MIDI information. This is complicated by the fact that it also has to identify the different types of sounds to assign them to different tracks.

The musical style recognition [7] paper deals with musical style recognition through supervised learning which is able to expose the genre specific attributes between the various genres selected for the dataset. The model being used for learning in this paper is through a back propagation neural net, with some innovative techniques. The data being used in this

training is musical data of different genres in different time periods. The neural network model defined in this paper is designed with three principles in mind : 1) musical input of varying length in an even manner 2) to model the hierarchical nature of musical recognition 3) and to capture the importance of directional flow in music. To achieve this, the concept of shared weights has been incorporated in this neural net, where similar edges have same shared weights, such as an edge from 1st beat to 2nd beat will be shared weight through the whole network. The shared weights technique also allows this model to deal with the issue of variable input size. Simply put, while the number of edges is allowed to vary with the size of the input, the number of edge types, and therefore the number of phrases, is constant. After running experiments on this model with about thousands of patterns, and 20 hours of learning, the accuracy was found to be 97% at an average.

Midicsv [8] tool performs the midi to notes rendering, where the input is taken as a .mid file, which is MIDI and outputs the CSV of notes which are present in the midi. This tool identifies all the notes that are playing, even if they are the overtones, and weren't meant to be recorded. The midicsv program allow you to use text processing tools to transform MIDI files or create them from scratch. midicsv converts a MIDI file into a Comma-Separated Value (CSV) text file, preserving all the information in the original MIDI file. The format of the CSV file is deliberately designed to facilitate processing by text manipulation tools.

The lstm approach to generate music as explained in [9] aims to achieve automatic piano music generation through long short term memory. The dataset being used here is the classical piano music midi dataset, which consists of MIDI files of classical piano pieces. RNNs are networks specialized in processing sequential data. With respect to other types of RNN, LSTM networks introduce a memory cell that allows to store old information to better capture long term dependencies. This paper tries to use a two layer LSTM, each layer containing 128 hidden nodes and a tanh function as the activation function, and the last layer being a softmax layer which gives the probabilities of the chords. Based on this, the model

has been trained, and to test it, they have performed it based on human experts, as the accuracy of music generation is very hard to obtain as music is an abstract concept.

In the audio recognition with recurrent neural networks [10], a audio chord recognition system is presented based on a recurrent neural network. The audio features are obtained from a deep neural network. For identifying the audio features, the paper has performed a spectral analysis and a principal component analysis to get the most diverse features. The features are run through a neural net with three types of edges, 1)single edges which represent the deterministic edges, 2)the dotted edges which represents the optional connections for temporal smoothing,3)the dashed edges which represent a prediction. Using this architecture, a maximum likelihood RNN model is trained. Following this, a beam search algorithm is used to find the most probable subsequences of a given time segment. This structure facilitates identifying the most promising paths by their cumulative log likelihood of all the previous nodes. Using a cross validation set using the mirex audio dataset, the RNN approach has provided 80.6% accuracy, whereas the other approaches have performed relatively poorly with around 60% accuracies.

Another novel approach to recognize chords in [11] uses the hidden markov model, trained by Expectation Maximization algorithm, and treated the chord labels as the hidden layer. In training the models, this paper has done a Baum welch algorithm to train the probabilities, giving only the chord sequence as input. The EM algorithm takes the observed feature vector, and computes the log likelihood of the feature vector given the hidden layer which is the chord, and the current model parameters. The main objective is to maximize the expectation which is the  $E[\log P(X, Q|\Theta)]$ . This specific application of EM to find maximum-likelihood parameter estimates for a hidden Markov model is known as the Baum-Welch, or forward-backward algorithm. The accuracy in percent they obtained was about 22% for recognition, respectively. The poor performance for recognition may be due to insufficient and unlabeled training data compared with a large set of classes (20 songs for 147 chord types).

Computoser [12] product aims to generate music based on the various filters provided, rather than accompanying the user in playing the piano piece. With the filters that are provided, such as the emotion, the instruments, classical, with chords, etc, the algorithm will generate appropriate music. Although this is a music generation tool, it performs chord generation if specified. This tool also generate arpeggios, which are broken chords, so as to give more randomness to the music that is being generated. There have been many songs that are automatically composed by this tool, and some of the top songs generated contain chord progressions, and sounds unique and new.

The chordal transitions in jazz approached in [13] paper aims to predict chord transitions for jazz pieces using three different types of models; on-line, off-line, and a hybrid. An on-line model is described as one where the learning happens on-the-fly. Initially, the chords being generated are random but eventually they converge to predict the actual chords itself by means of some feedback mechanism. Here, each song is its own test set. An off-line model aims to learn from a carefully manufactured dataset, in this case 49 jazz pieces. Results show that on-line models are on par with the off-line ones even though they work lesser amount of data. This goes to show that even data that is localized to a song (such as a few bars of data) can hold lots of information, and is a good indicator for prediction. The paper also discusses the common pitfalls in music analysis, such as the problems in measurement of accuracy, representation of datasets, and algorithmic restrictions imposed by jazz. In closing, the paper describes the hybrid model which is initialized with off-line learning but proceeds to learn on-the-fly. This model has an accuracy of around 53%.

Bach harmony [14] a dataset present on the UCI Machine Learning Repository, which has been cited by multiple papers. The dataset has over 5000 instances, where each row indicates presence of each pitch class right from C through B. These are represented as twelve boolean columns, each column representing a pitch class. Finally, the last column is a chord label which has been manually annotated by a human expert. These harmonies occur in the

chorales written by Bach, and the dataset references the chorale in which the harmony occurred. The chorales dataset is also available in the same UCI repository.

Aubio is a tool that aids extraction of information from any audio file (that represents music). It is written in C and has no external dependencies. It can also interface well with Python's numpy, where the C arrays can directly be viewed as numpy arrays. This makes the implementation quite efficient, and fast as well. Some of the features aubio provides are :

- Onset Extraction : To detect when discrete sound events begin in an audio signal
- Pitch Extraction : To detect the exact pitch of a note
- Tempo Extraction : To detect the number of beats per minute

The hybrid system for automatic generation [16] describes a three-step procedure to generate bass accompaniment for any melody. The first step is to identify the so called chord tones in the melody. Each bar of the melody is most likely to have the root note of the chord being played in the bass. This note is identified by means of a Support Vector Machine, and bars where the true chord tone is unambiguous is marked as a checkpoint. The next step is to generate all possible chord progressions to get from one checkpoint to another. This is achieved by means of Neo-Riemannian transformations, which is a set of rules defined by music theory to encapsulate the concept of chord progressions. Finally, a Markov Process is used to evaluate probabilities of all progressions, which are stored in a tree like data structure, and the best combination is chosen. The training is done only on three songs of a British band named Radiohead, and the algorithm is tested on the next two songs. The actual accompaniment in the original song is taken as the ground truth, and the results of the algorithm are compared against that. The results are promising with the algorithm being able to generate 81% of the chords correctly after having learnt from only three songs. However, the paper also claims that increasing the size of the dataset might lead to over-generalization of chord patterns and would diminish the performance of the system.

Chord recognition from an audio signal is a form of information retrieval which can be used as a precursor to various tasks such as music emotion classification. Other forms of information retrieval include extracting MFCC (Mel Frequency Cepstral Coefficients) which are low level features which can be extracted from an audio signal. This paper [17] aims to extract mid-level features which can be used to infer more higher level features of the audio. It uses Language Modelling techniques by treating each chord as a word and a chord sequence as a sentence. Here, an n-gram model fits best in order to predict the next chord in the sequence. The model achieves ~ 70% accuracy on the test samples. Further, the paper also describes classification of songs based on emotion using features such as Chord Histogram and Longest Common Chord Subsequence. This classifier yields 60% accuracy.

Bachbot is a tool developed by researchers at Cambridge which aims to recreate Johann Sebastian Bach's chorales. The project is an effort to achieve Computational Creativity and bridge the gap on what AI cannot do. The website for the tool offers a Turing Test by displaying two separate music pieces, one which is an original Bach chorale, and the other which was generated by BachBot. As a form of normalization, all songs are converted to C Major scale and run at 120 beats per minute. The tool has also been used to superimpose chorales over pre-existing music, and the results are pleasant to the human ear.

Another attempt to generate harmonies with the use of LSTMs[19] uses the concept of absolute time in feeding data to the model. Dealing with MIDI files for data, it learns long-term autocorrelations in the music that are aligned with the tempo of the music piece. The model learns long term aspects of the song style such as the chord structure and is used to generate jazz music, which is subjectively perceived as pleasant according to the authors.

With regards to prediction of chords, there exists a dissertation[20] on the subject of a jam support system which includes an implementation of using previous bar and chord information to predict upcoming chords. The model elaborated utilizes a combination of a Hidden Markov Model and a Variable Order Markov Model. The Hidden Markov Model gets

the hidden chord states for the observed notes. The Variable Order Markov Model uses the sequence extracted and divides into a set of subsequences. It uses the next chord information on each subsequence and aggregates the data to make a prediction on the whole information extracted. The accuracy of the system on 7 labels comes to 42 percent.

## CHAPTER-5

### PROJECT REQUIREMENTS SPECIFICATION

#### 5.1 Modules

##### 5.1.1 Input module

This module takes the input from the user as midi format, from the keys being played by him on the keyboard. This will interface with the other modules to predict what the left hand chord could be. This module creates the shared memory segment if it doesn't exist, and pushes the predicted chords into it for output module to consume.

##### 5.1.2 Model module

This module takes in the input that was obtained from the input module, and runs one of the models as selected by the user, to predict what the left hand sequence of notes could be. There are plenty of sub modules implemented here which correspond to the various models.

##### 5.1.3 Output module

This module takes the predicted chord from the model module, and it outputs the sequence of notes that have to be played to the keyboard. The output module interacts with the GUI and lights up the buttons that are being played to the keyboard.



#### 5.1.4 Generator module

This module generates the music, both the left and the right hand notes and generates some kind of melody. This module exists by itself and it is an n-gram model.

#### 5.1.4 GUI module

This module is the entry point of the product. The GUI is coded mainly in tkinter, which starts off with a UI for the user to enter the tempo she is going to play the composition in. After she has entered the tempo, a keyboard will be displayed to the user, which denotes the left hand notes that is being generated by the underlying prediction. The user can look at the keyboard layout, and see what are the chords being predicted.

### 5.2 Constraints

This product is essentially a tool that is plugged in a desktop or a laptop, which is connected to the keyboard the user is playing on.

#### 5.2.1 Design Constraints

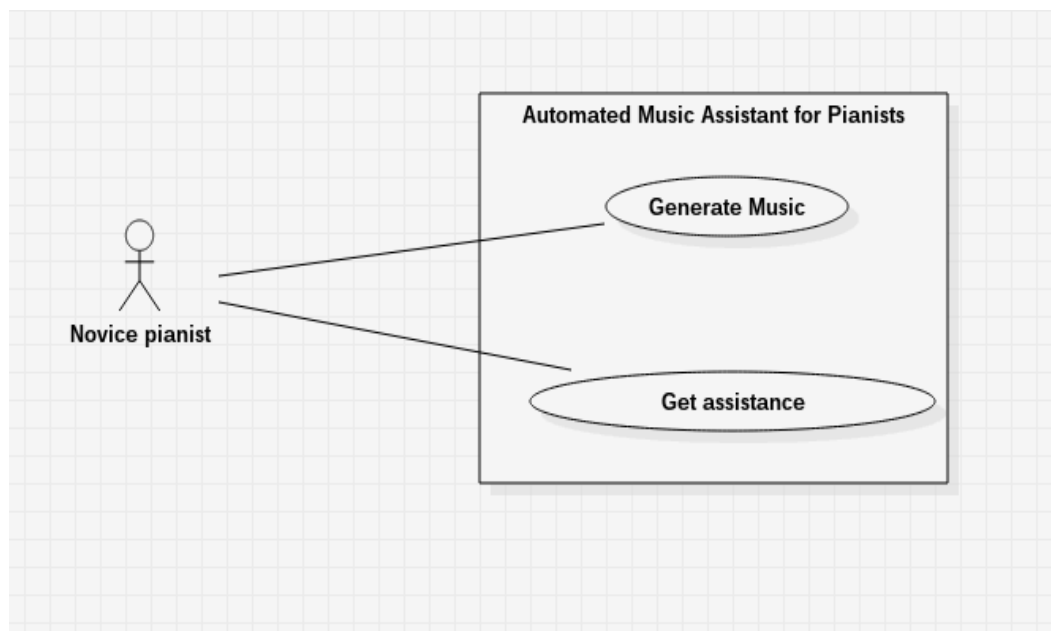
- In terms of design, this product is mainly aimed at rookie musicians who want to develop their skill by first playing easy songs. Hence, the complex songs is not learnt in the scope of this product.

- As this product is aimed at beginners, there is a simple assumption from a design perspective that the barrier is the middle C in the keyboard. Every note to the left of it is played by the left hand, and every note to the right of it is played by the right hand.
- There are two parallel operations that are running simultaneously in our product are the two hands, which is reading the treble notes from the user, and processing that, and the other process, which renders the processed bass notes on the keyboard through the midi connector.

### 5.2.2 System Constraints

- Since this is a realtime product where the notes are being played as soon as the user plays it, the latency should be minimum. As latency is a relative metric, we would have varying values for latency spanning across different machines.
- Communication interfaces include the interface to the piano from the laptop, which is the midi connector. This is a midi cable which is connected on both ends to the laptop as well as the piano.

## 5.3 Use Case Diagram

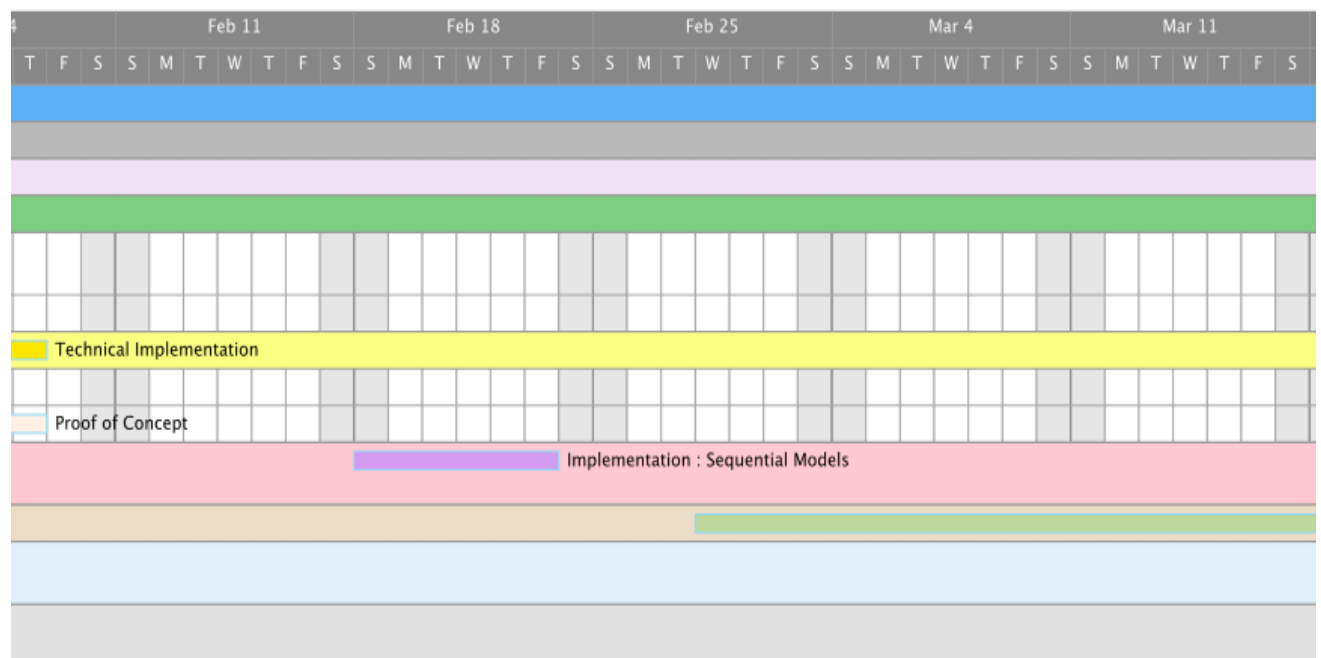
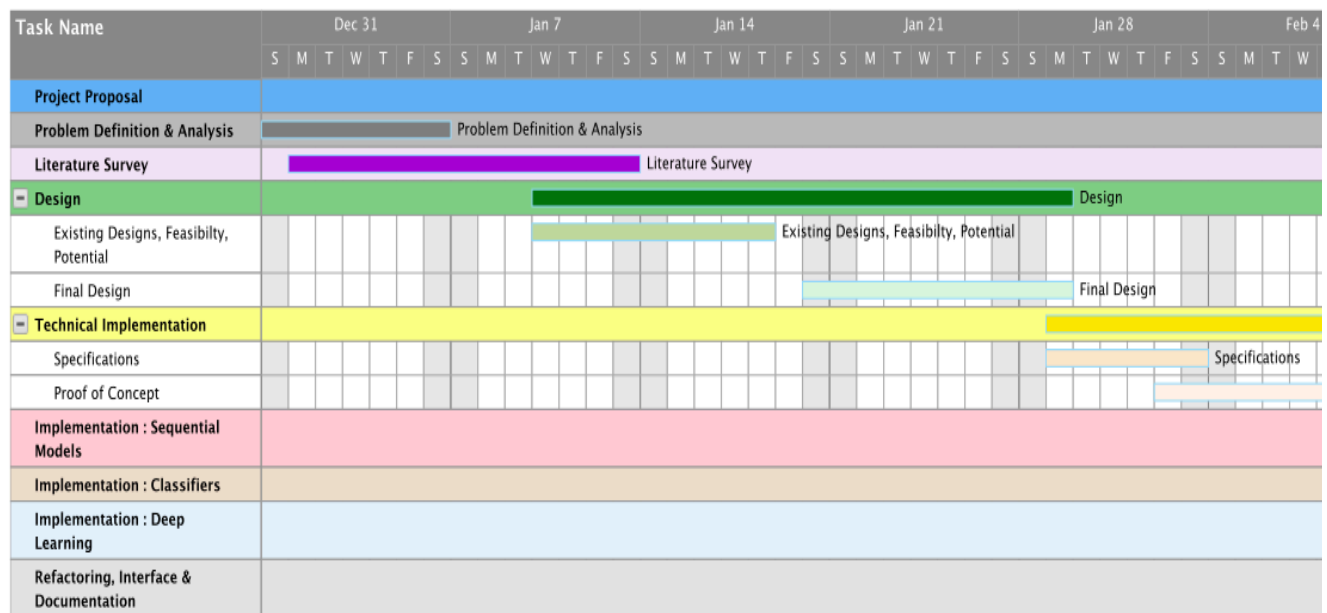


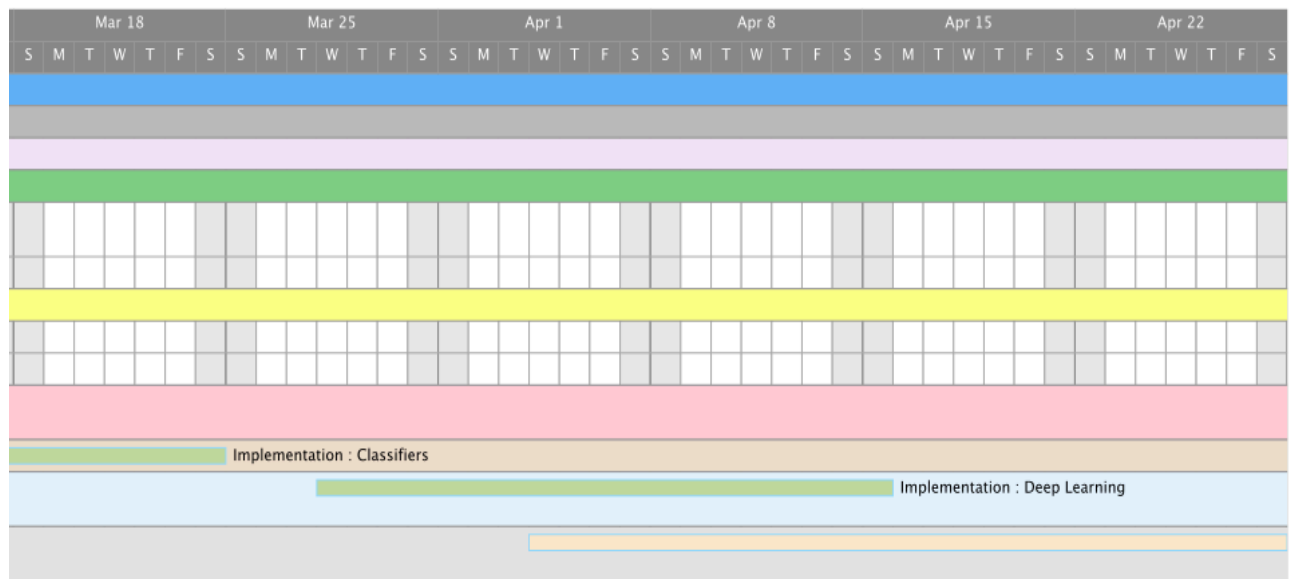
**Fig. 5.1 Use Case Diagram**

Use Case Item	Description
Generate music	This use case is pure generation, and a user can generate both the right hand as well as the left hand melodies and play them together
Sound better with associated melody	This use case is related to how the user interacts with our system, and sounds better with our generated chords.
Learn different ways of generating bass	The user could also learn different ways to generate bass/left hand chords.

**Table 5.1 Use Cases**

## 5.4 Gantt Chart





**Fig. 5.2 Gantt Chart**

## 5.5 User Characteristics

The eventual users of this product will be beginner keyboard players, who are struggling to play with both their hands at the same time. The targeted audience are music schools where they would want to help their students learn as quickly. Also, it can be used by beginner pianists to sound better, and focus on one skill at a time, which is the treble notes. As our targeted audience are beginners who have just started to master the art, they might find it hard to synchronise their left hand and right hand while playing, hence our product will aim to make them sound better, and at the same time help them to learn.

As this is aimed at beginners, the requirements will be specific to them, and we won't be dealing with complicated melodies, where the bass chords will overlap with the melody. In these kind of piano pieces, there is a clear distinction between the bass and the treble notes.

## 5.6 Risks

### 5.6.1 Data Requirements :

As this is a Machine Learning based solution, an inherent requirement is a reliable dataset. A quick online survey reveals that there are no such sources on the internet, in the format the algorithm requires (MIDI). The algorithm cannot learn to produce harmonies without a training dataset, and obviously, there cannot be a validation score without a testing dataset.

### 5.6.2 Hardware Latency :

In case the computations are too complex, a real-time solution as proposed here might fail to produce results on time. Unrealistic expectations from the user's hardware might lead to failure of the product.

### 5.6.3 Measure of Quality :

As music is an abstract entity, evaluation of the results cannot be done using a metric acceptable to all. The product might not be well received amongst the users who do not like what the results sound like, whereas the same results could be highly appreciated by another set of users.

### 5.6.4 Time :

As this is part of an academic project with a short timeline, implementation of all features in a robust manner might prove to be a challenge.

## CHAPTER-6

### SYSTEM REQUIREMENTS SPECIFICATION

#### 6.1 Functional Requirements

##### 6.1.1 Generate Music

The system shall generate both the left and the right hand notes, which is being learnt from the data that has been provided to it. Either one song, or many songs can be provided to the system, and the underlying distribution is being learnt. The underlying distribution that is defined by the data, is captured with the help of the models that is listed in the model module in the previous section. This requirement is satisfied by performing models, which in turn uses the concept of ngrams to understand the distribution.

##### 6.1.2 Assist the user

The user of our system will interact with the input module, and the system will generate the left hand chords which will accompany the right hand notes being played by the user in real time. The model to generate the left hand chords is tweakable, and can be specified by the user. The chords that are being played are not only

#### 6.2 Non-Functional Requirements

##### 6.2.1 Performance

From performance perspective, this product has to be extremely fast, as the latency must be minimised. This has to be a requirement as it is a realtime product. The latency expected must be around 1ms.

### 6.2.2 Manageability

From manageability point of view, this product is pretty manageable as it is a standalone product that is deployed on the end user's machine.

### 6.2.3 Reliability

The reliability of the overall system depends on the reliability of the separate components. The main part of reliability is the model which is stored in the user's system.

### 6.2.4 Maintainability

The system is maintained only in terms of patches from the software side, which has to be installed and updated in the client software.

## 6.3 Hardware Requirements

- A laptop PC with at least 4 GB of RAM is the minimum requirement to use the product, as there would be a fair amount of computation at runtime.
- The product will support all keyboards which have a MIDI port, and are capable of transmitting real-time MIDI information to a PC. This is required as this is the interface to transmit midi data from the keyboard to the laptop and vice versa.

## 6.4 Software Requirements

- The only software limitations for this product is the python programming language, and the pygame module for the connector with piano for the midi transfer. This product would work irrespective of the operating system it is run on.



- Apart from this, the other requirements that are present from the software side are listed in the table below.

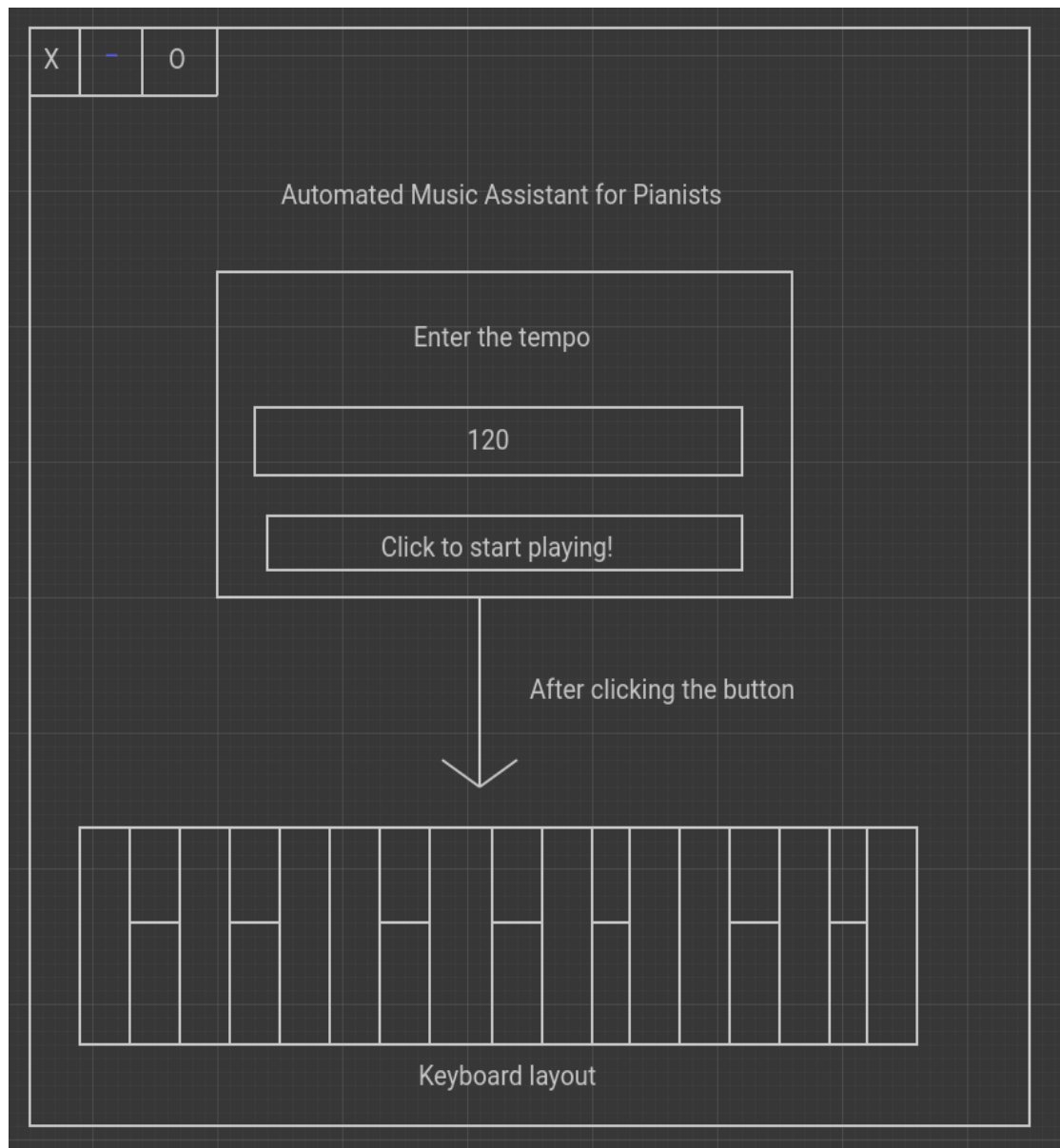
Name	Version	Source
Python	>3.4	<a href="https://www.python.org/downloads/release/python-364/">https://www.python.org/downloads/release/python-364/</a>
Pygame	1.9.2	<a href="https://www.pygame.org/docs/ref/midi.html">https://www.pygame.org/docs/ref/midi.html</a>
scikit-learn	0.19.1	<a href="http://scikit-learn.org/stable/install.html">http://scikit-learn.org/stable/install.html</a>
numpy	1.11.2	<a href="https://sourceforge.net/projects/numpy/">https://sourceforge.net/projects/numpy/</a>

**Table 6.1 Software Requirements**

## 6.5 Communication Interfaces

A MIDI cable is the primary communication interface that is required. The product will not use an internet connection to function, and this no other networking protocols are required to be followed. Once a MIDI connection between the keyboard and the laptop is established, the product can be used.

## 6.6 User Interfaces

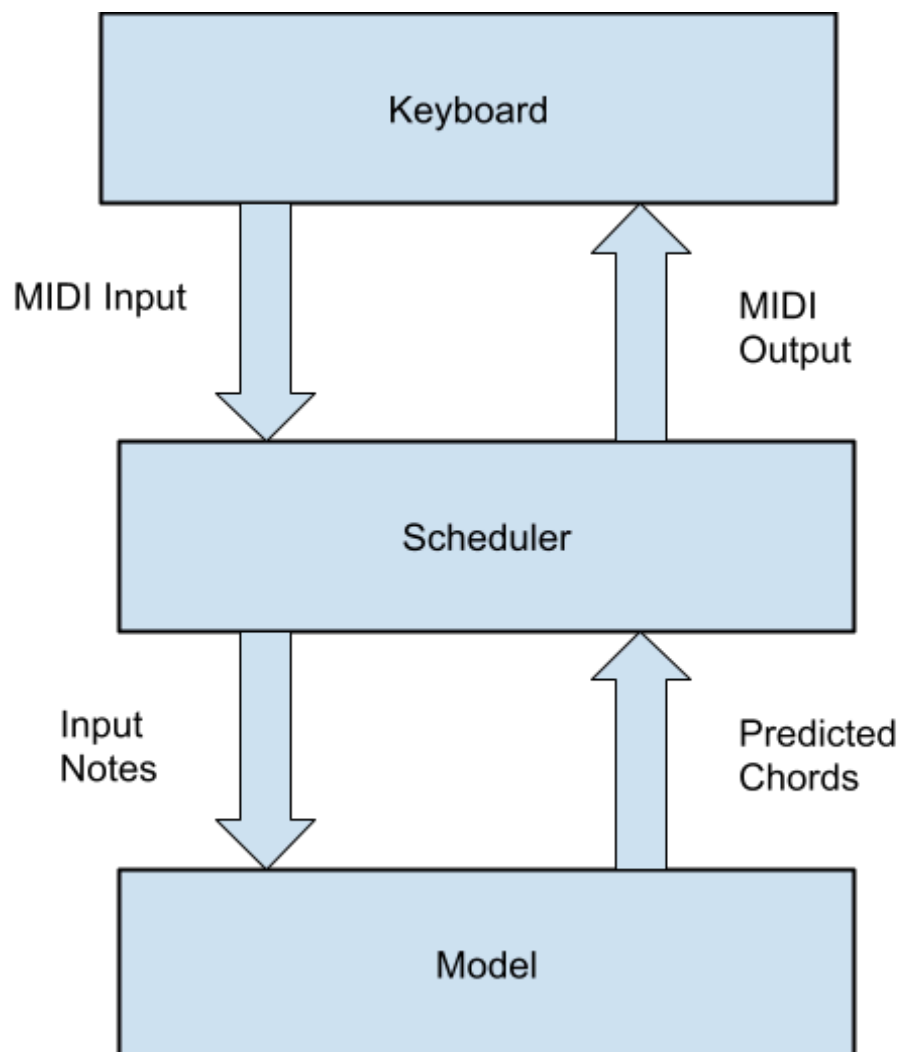


**Fig. 6.1 User Interface**

The product will have a very simple graphical user interface due to the short timeline. The UI will prompt the user to select a model to generate chords, and will also take as input the tempo at which the user is going to play. Upon clicking 'Play' the UI will trigger scripts on the backend which will communicate with the keyboard (expected to be connected while using the UI) and enable a harmonious outcome.

## CHAPTER-7

### SYSTEM DESIGN



**Fig. 7.1 System Design**

AMAP system utilizes 3 components at the highest level -

- The Keyboard
- The Scheduler
- The Model

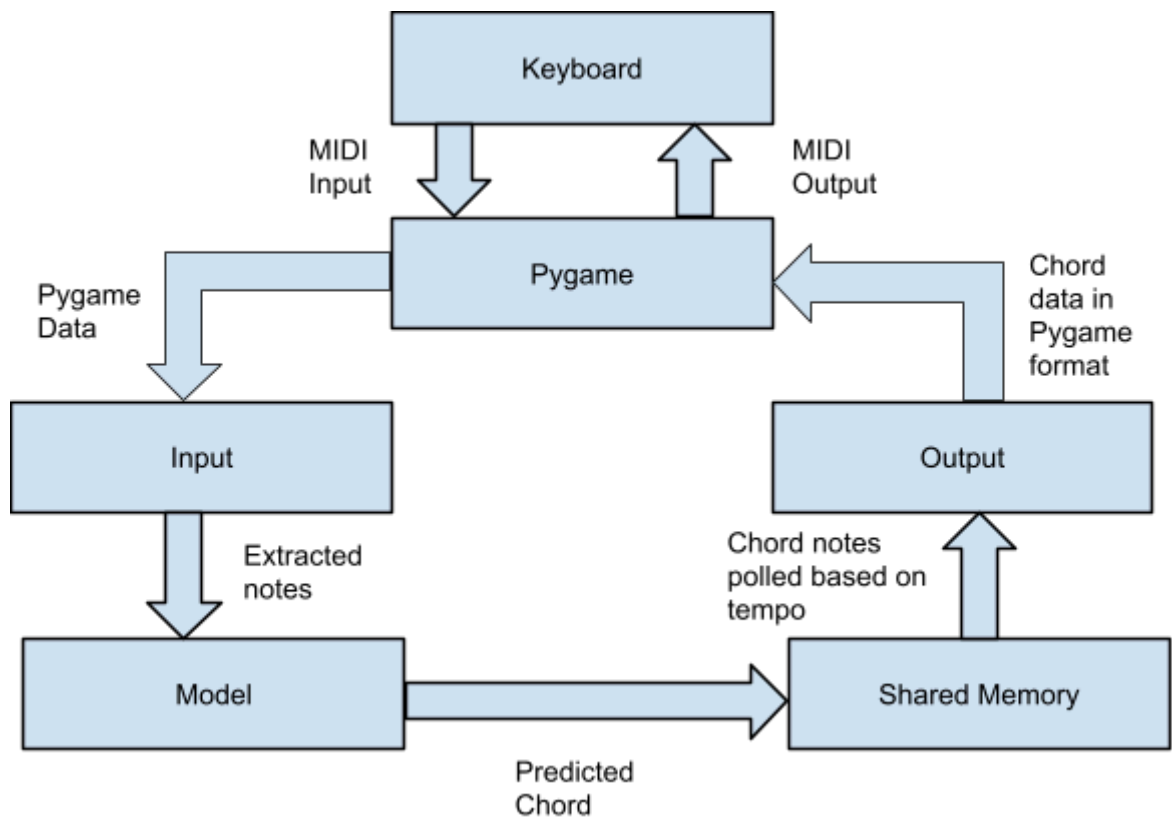
The Keyboard is the end interface used by the customer. He/She plays the notes on the keyboard which are then transmitted to the AMAP system through the MIDI Port and processed there.

The Model is a machine learning model that takes in the user input and predicts chords to be played. What is to be predicted is learnt using a significant amount of training data, on which the model is fitted and from which it learns. The predicted chord is sent back to the Scheduler which processes it further.

The Scheduler is the interface between the Keyboard and the model. It continues to poll, waiting for an input from the keyboard, then once the notes played from the keyboard are obtained, transmits this information to the model for processing. The output from the model is obtained at time intervals configured by parameters such as tempo among others, and then the chords and arpeggios to be played are transmitted to the keyboard through the MIDI port, where it is played.

## CHAPTER-8

### DETAILED DESIGN



**Fig. 8.1 Detailed Diagram**

### 8.1 Module specifications

#### 1. Keyboard -

Input: User keystrokes

Output: Audio sound

Description: As mentioned in the System Design, the keyboard is the end interface through which the customer interacts with the AMAP system. The MIDI ports are the connection between the keyboard and the system and are used to transmit note and chord data to and from the system.

## 2. Pygame -

Input : MIDI data/Integer note values

Output : Integer note values/MIDI data

Description : The Pygame module is a library used to interface MIDI input/output with Python code. During the input phase, it reads notes from the keyboard and outputs a list containing the following information in this structure - [[statuscode, note value, velocity], timestamp]. This is received by the interfacing python modules that extract required data. During the output phase, it uses the functions note\_on and note\_off to tell the keyboard when to start and stop playing a given note. The parameters given to these functions are the note value, and the velocity with which it is played.

## 3. Input -

Input : Pygame MIDI input data

Output : Extracted notes

Description: This is a process that polls the Pygame input stream continuously and extracts note data when it is written there. The note values are then passed on to the next module for processing.

#### 4. Model -

Input : Input Notes

Output : Predicted Chord

Description : The model is a machine learning model that takes in input information of the last several bars played on the keyboard and, using this, predicts the chord that is to be played. The model is fitted and trained using a dataset for learning such patterns between notes and chords.

#### 5. Shared Memory-

Input : Note data

Output : Note data

Description : A shared memory location is required for interfacing between the input and output process and the model. The output of the model is stored in this cache which is queried by the output process.

#### 6. Output -

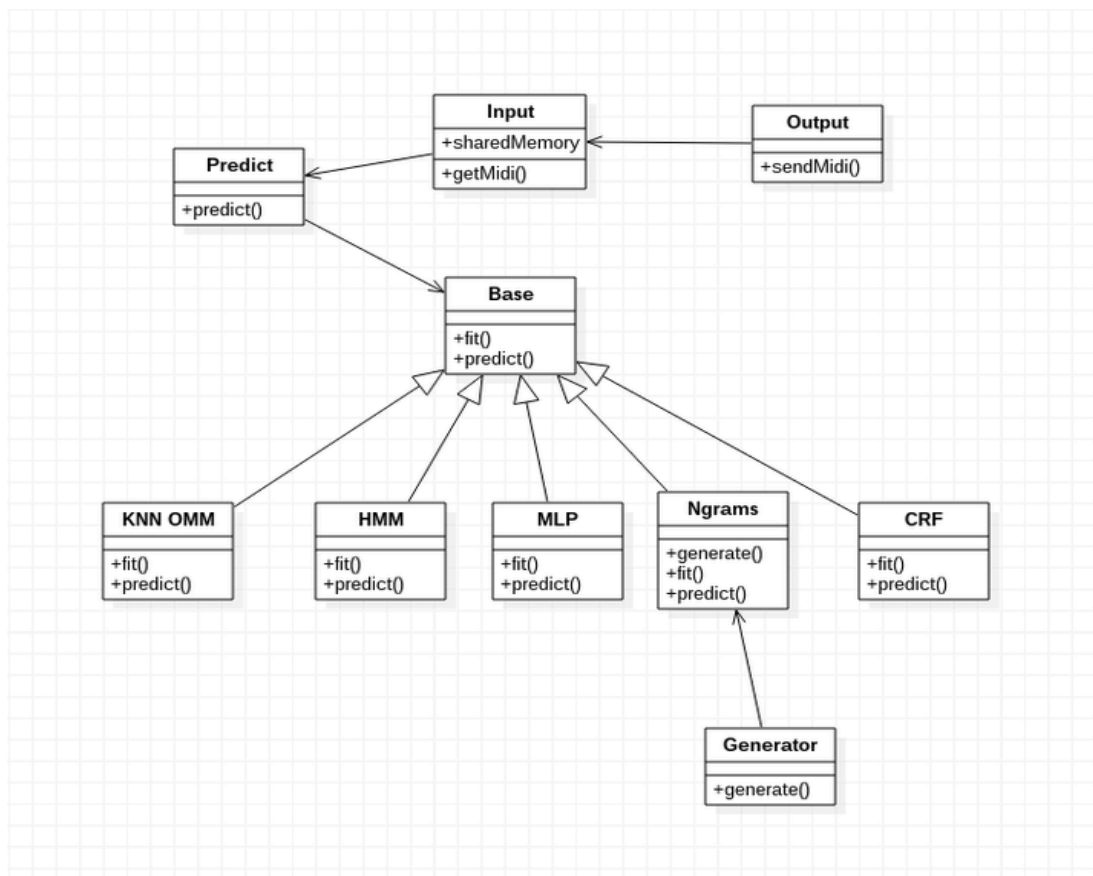
Input : Chord notes

Output : Pygame formatted note data

Description : The Output process queries the shared memory continuously, and when a note is present, consumes it from the cache and converts into the Pygame format. It sends the data to the pygame module in time intervals determined by the tempo.



## 8.2 Master Class Diagram



**Fig. 8.2 Master Class Diagram**

In terms of a class perspective, AMAP is divided into three major classes Input, Output, and Model where Model is subclassed into several subtypes on the basis of different algorithms implemented to perform the task of chord prediction.

### 8.2.1 Class specifications

## I. Input

### 1. Attributes:

- a) `shared_memory`- This is a reference to the Shared Memory instance used in order to populate the predicted chords returned from the model to be played on the keyboard

### 2. Methods:

- a) `push_chord(bar_notes) -> predicted_chords` : This method takes in the `bar_notes` from the Pygame input stream and feed it to the model to get the chords to be played back on the keyboard.

## II. Output

### 1. Attributes:

- a) `shared_memory` - This is a reference to the Shared Memory instance that is queried for chords being populated by Input

### 2. Methods:

- a) `send_output()` : The `send_output` function constantly queries the `shared_memory` reference for new chords. When it finds such, it transmits them to the Pygame module on the basis of configured tempo. When a note is sent to Pygame it is immediately played on the keyboard

## III. Models

### A. Model:

#### 1. Attributes: NA

#### 2. Methods:

- a) `__init__(self)`:
- b) `fit` : This is the abstract method `fit`, and every model class must override this method.
- c) `predict` : This is an abstract method as well, which takes an input X vector, and produces a output Y corresponding to it.

## B. KNN:

### 1. Attributes:

- a) `clf` : This is a simple classifier that depicts the KNN algorithm. It must be trained on inputs that is given through the fit function.

### 2. Methods:

- a) `__init__(self)`:
- b) `fit(bar_sequences, chord_sequences)` : The fit function takes as input the bar sequences and chord sequences and uses the K-nearest neighbours algorithm to learn mappings between bars and chords
- c) `predict` : The predict function takes a bar sequence, and provides a chord label that is predicted by the model.

## C. OMM:

### 1. Attributes:

- a) `clf` : This is a simple classifier that depicts the OMM algorithm. It must be trained on inputs that is given through the fit function. It learns the pi matrix as well as the transition matrix.

### 2. Methods:

- a) `__init__(self)`:
- b) `fit(chord_sequences)`: The fit function takes a series of chord sequences, and learns the pi matrix as well as the transition matrix of a simple markov chain model. There is no need of the bar sequences here, as it is trained only on the chord sequences.
- c) `predict(chord)` : Given a chord, this model provides the next most probable chord in a sequence. This denotes the concept of chord progression in music. This added with the KNN model gives us the model to perform real time chord prediction.

## D. KNN-OMM:

### 1. Attributes:

- a) `clf`: This represents the reference to the classifier instance to be initialized and used for prediction
- b) `knn`: reference to initialized KNN object instance
- c) `omm`: reference to initialized OMM object instance
- d) `ngramlength`: This specifies the length of the ngram in terms of the number of previous bars to be taken into account as data
- e) `chords_in_ngram`: boolean determining whether previous chords to be learnt as well
- f) `num_notes`: number of notes to be taken from the bar (from the beginning)

### 2. Methods:

- a) `__init__(self, ngramlength, chords_in_ngram, num_notes)`:
- b) `fit(bar_sequences, chord_sequences)`: This model directly utilizes the KNN and OMM models. It contains a reference to an instance of both models and uses them for training. The KNN model is fit on the `bar_sequence` and `chord_sequences` while the OMM model is fit on the `chord_sequences` only.
- c) `predict(bar_sequence)`: The predict function utilized both the KNN and OMM algorithms to perform its task. It takes a given `bar_sequence` and passes it to the KNN model to predict a chord label mapping to that bar. It then takes this chord label and passes it to the OMM model which predicts the next chord label in reference to the one passed to it. In this manner, the KNN-OMM model takes in the previous bar played and predicts the current chord to be outputted.

## E. HMM:

### 1. Attributes:

- a) `clf`: This represents the reference to the classifier instance to be initialized and used for prediction
- b) `data_type`: This is a hyperparameter used to configure the type of data to be extracted and then used for learning
- c) `ngramlength`: This specifies the length of the ngram in terms of the number of previous bars to be taken into account as data
- d) `num_notes`: number of notes to be taken from the bar (from the beginning)

## 2. Methods:

- a) `__init__(self, data_type, ngramlength, num_notes)`:
- b) `fit(bar_sequences, chord_sequences)`: The HMM model takes in the `bar_sequences` and `chord_sequences` and uses them to create the three matrices based on which the algorithm the model uses - The Viterbi Algorithm - is based on. It uses only the chord sequences to generate the start probability vector and the transition matrix. The start probability vector indicates the probability of a given chord being played at the very beginning. The transition matrix associates a value between two given chord labels for each  $i,j$  index in the matrix - this being the probability that the  $j$ th chord will be played given that the  $i$ th chord was played previously. Third is the emission matrix, which gives the probability that a bar is played given that a chord is played in the same time frame. Thus the model considers the chord to be the hidden state and the bar sequence to be the observed state. As the chord is the label to be predicted, it is appropriate for it to be the hidden state, rather than the bar sequence.
- c) `predict(bar_sequence)`: The model takes in a sequence of bar notes and uses the matrices learned by the fit function to predict

a series of chord labels for each note in the sequence. The last note of the sequence is taken to be the predicted label for the current bar.

#### F. Logistic Regression:

##### 1. Attributes:

- a) `clf`: This represents the reference to the classifier instance to be initialized and used for prediction
- b) `data_type`: This is a hyperparameter used to configure the type of data to be extracted and then used for learning
- c) `activation`: The type of activation function to be used
- d) `ngramlength`: This specifies the length of the ngram in terms of the number of previous bars to be taken into account as data
- e) `chords_in_ngram`: boolean determining whether previous chords to be learnt as well
- f) `num_notes`: number of notes to be taken from the bar (from the beginning)
- g) `softmax`: boolean determining addition of a softmax layer

##### 2. Methods:

- a) `__init__(self, data_type, activation, ngramlength, chords_in_ngram, num_notes, softmax)`:
- b) `fit(bar_sequences, chord_sequences)`: This function uses the logistic regression algorithm with 'lbfgs' solver to map the `bar_sequences` inputted to the appropriate chord labels in `chord_sequences`
- c) `predict(bar_sequence)`: The predict function queries the fitted model with the passed `bar_sequence` parameter and returns the predicted chord label

#### G. Multilayer Perceptron:

##### 1. Attributes:

- a) `clf`: This represents the reference to the classifier instance to be initialized and used for prediction
- b) `data_type`: This is a hyperparameter used to configure the type of data to be extracted and then used for learning
- c) `activation`: The type of activation function to be used
- d) `ngramlength`: This specifies the length of the ngram in terms of the number of previous bars to be taken into account as data
- e) `chords_in_ngram`: boolean determining whether previous chords to be learnt as well
- f) `num_notes`: number of notes to be taken from the bar (from the beginning)
- g) `softmax`: boolean determining addition of a softmax layer

## 2. Methods:

- a) `__init__(self, data_type, activation, ngramlength, chords_in_ngram, num_notes, softmax)`:
- b) `fit(bar_sequences, chord_sequences)`: This function uses a Multi-Layer Perceptron, which is a fully connected Neural Network taking in bar\_sequence notes and passing it through a hidden layer to map it to one of the 7 chord outputs. The mapping between a uniform length padded bar sequence and the chord label is learnt in this function
- c) `predict(bar_sequence)`: This function uses the saved Neural Network MLP model to predict the chord label of the bar sequence passed to the function

## H. SVM:

### 1. Attributes:

- a) `clf`: This represents the reference to the classifier instance to be initialized and used for prediction

- b) `data_type`: This is a hyperparameter used to configure the type of data to be extracted and then used for learning
- c) `activation`: The type of activation function to be used
- d) `ngramlength`: This specifies the length of the ngram in terms of the number of previous bars to be taken into account as data
- e) `chords_in_ngram`: boolean determining whether previous chords to be learnt as well
- f) `num_notes`: number of notes to be taken from the bar (from the beginning)
- g) `softmax`: boolean determining addition of a softmax layer

## 2. Methods

- a) `__init__(self, data_type, activation, ngramlength, chords_in_ngram, num_notes, softmax)`:
- b) `fit(bar_sequences, chord_sequences)`: This function uses a support vector algorithm that takes in bar sequences and chord sequences and learns the hyperplanes associated with the output labels.
- c) `predict(bar_sequence)`: The predict function predicts the output labels by mapping the bar sequences notes to the output plane and extracting the output label associated with the region it is mapped to.

## I. LSTM:

### 1. Attributes:

- a) `model` : This is the model that depicts the entire LSTM architecture. The architecture is initially a sequential layer, which is then followed by a LSTM layer containing 50 cells. These cells are of the dimension  $\text{timesteps} * \text{features}$ .

### 2. Methods:



- a) fit : This method takes a set of X values, and a set of Y values, and fits it to the architecture as defined by the constructor of the LSTM class. It gets an input, and reshapes it to how the LSTM can consume it, and then fits it.
- b) predict : This method takes a X value, which should be consumable by the LSTM model, and it then passes it through a softmax layer, which then produces the output as a bunch of probabilities. From these probabilities, the argmax function is called to return the index of the maximum probability.

#### J. CNN:

##### 1. Attributes:

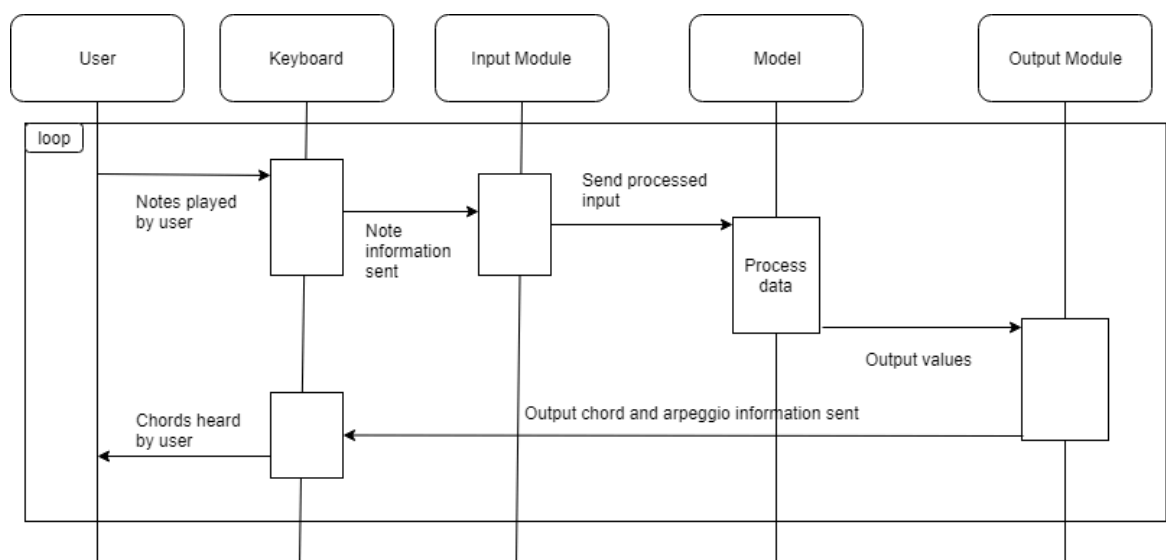
- a) clf : This is an object of the Sequential class of the keras deep learning module. Using the .add function, additional layers of neurons can be added to this classifier object which represents the Convolutional Neural Network.
- b) epochs : This value indicates the number of epochs to run the training for. Larger number of epochs could lead to overfitting of the dataset and thus poor performance on test.

##### 2. Methods

- a) Fit : This method is used to learn the weights of the neural network, i.e to train the network and obtain the hyperplane. This is performed by means of back-propagation where the error from the output layer is propagated backward to train the weights of the neurons or filters.
- b) Predict : This method takes a sequence of bars represented in the image format, and runs it through the neural network to obtain the prediction or classification for the instance.
- c) Score : The score method is used to test the performance of the model, and returns the percentage accuracy of the predictions

made for a given set of bars. It takes both bars and their corresponding chords as parameters.

### 8.3 Sequence Diagram



**Fig. 8.3 Sequence Diagram**

This sequence diagrams describes the whole process of using our product, and the interfacing with the user as well. First, the user interacts with the keyboard, which then interacts with the input module of our product, which consumes the stream of notes coming in, and passes it to the model, which then predicts a chord based on the algorithm, and is output from the output module. It can be thought of a stream of HTTP requests, where there is a stream coming into

the system, and the system consumes the requests, and outputs the responses from it. In a distributed architecture, it can be thought of a component communicating with other components with the help of a message bus, where there are events being published. Here the events are being published by the keyboard module, and is consumed by the model module, and an event is sent back to the keyboard module with the help of the output module.

## CHAPTER-9

### IMPLEMENTATION AND PSEUDO-CODE

The input and output modules are responsible for reading and writing music from and to the keyboard device, and are highly dependent on the MIDI interfaces provided.

#### Input

```
initialise_shm(notes)
attach_shm(notes)
while True:
    bar = read_note_stream_from_keyboard()
    chord = predict_chord(bar)
    write_to_shm(chord)
```

#### Output

```
attach_shm(notes)
while chord in SHM:
    chord = read_chord_from_shm()
    play_chord(chord)
```

They work in tandem and read and write simultaneously. A shared memory segment is used as a means to communicate between two processes. The requirement for two processes arises from the fact that both input and output need to happen in parallel and this cannot be achieved with one process.

With respect to the functionality of complementing the user, the crux of all the models implemented is represented via an abstract base class.

### Generator

```
composition = get_composition(3) #trigram
while melody,chords in composition:
    play_melody(melody)
    play_chord(chord)
def get_composition(n):
    ngram_model = get_model()
    return generate_composition(ngram_model)
```

The generator module is for generating new music. The model that is being used for generating new music is the ngram model. The ngram model trains the data on the processed data, and provides the model in terms of ngram probabilities back to generator. The generator module generates the composition by taking the ngram probabilities and generates a sequence of music ngrams. This achieves the context sensitivity of music, and generates a good composition.

The implementation of the input and output has been done in python with the help of pygame library, as it has features to connect to the keyboard with the help of a MIDI interface.

```
class Base(ABC):
    def __init__(self):
        pass
    @abstractmethod
    def fit(self, x, y):
```

```
        pass
    @abstractmethod
    def predict(self, x):
        pass
    def score(self, x, y):
        pass
```

The base class defines an interface comprising of three methods (fit, predict, and score) along with a constructor. Each model developed as part of the project inherits from the base class and implements the methods defined here. The interface is inspired by the scikit-learn Python library which uses similar notation for Machine Learning modules.

1. `__init__` : This is the name of the constructor and returns an object of the class when called. It takes no explicit parameters in the base class, but sub-classes may take values to initialize the hyperparameters of the Machine Learning model. Eg : Activation unit, number of epochs, or kernel.
2. `fit` : This is the function where the learning aspect of the model is implemented. It takes as parameters X and y, which are the independent and dependent variables. In the context of the project, X is a sequence of bars of music while y is the corresponding sequence of chords that were played with the left hand in order to complement the right hand.
3. `predict` : The predict function is responsible for providing a prediction label when given an instance of data. In the context of the project, it would accept a bar of music as a parameter and return a string which corresponds to a chord label.
4. `score` : The purpose of the score function is to evaluate how well the model has learnt the concept. The function accepts both X and y, like the fit function, but internally invokes

predict on each of the Xs. The predictions are compared with the true labels and the percentage match is returned.'

The implementation of the GUI is mainly done using the tkinter library. This will be the entry point for the user if he wishes to use the application.

### GUI

Create the tempo UI.

while tempo not entered:

    wait

close the tempo UI

create the keyboard layout

while window not closed:

    wait for the notes to be played

    if note played:

        light the note in keyboard UI

        sleep interval

        revert to original colour

This pseudo code that is displayed above is used to implement the GUI. The GUI module interacts with the output module, and displays the notes in the keyboard.

## CHAPTER-10

### TESTING

There are two types of testing performed in this project.

1. Software Testing : To verify correctness of code functionality
2. Model Testing : To check accuracy of ML models

**Software Testing** : A vast majority of the functions written are meant to perform data processing and manipulation tasks on the music csv files. It is vital to ensure that these functions are correct because the repercussions of them failing to produce correct results are massive. Wrongly formatted data passed to the models would lead to incorrect learning and thus poor results. A set of unit tests and sanity checks were performed on certain central functions in the code.

`sequence_vectors` :

- input : csv file paths
- output : (bar sequences, chord sequences)

`parse_data` :

- input : csv file paths, extra parameters for specific formats of data
- output : (bar sequences, chord sequences)

`create_image_from_bar` :

- input : bar notes, bar length
- output : matrix which portrays the bar as an image (for CNN)

`batch_format.py` :

- input : raw files of music data
- output : processed files which is in the project-specific .formatted format.



**Model Testing :** The project involves development of multiple ML models such as Support Vector Machines, Multi Layer Perceptron, Logistic Regression, etc

These models must be tested for their accuracies. This is accomplished by means of a yaml file which encapsulates various configurations of every model. A script then queries the yaml and trains each model specified and calls the score method of the model to produce results in terms of accuracy.

The data is split into three sets, namely Train, Test, and Validation.

**Train :** This consists of about 75% of the complete dataset. This is used for the training the model, and is treated like the primary dataset.

**Validation :** This consists of 25% of the complete dataset. The purpose of the validation set is to run the model trained of the Train set and fine tune it to improve the accuracy on unseen data.

**Test :** This is the remaining 25% of the dataset. This is where the final accuracy of the model is derived from and it is the equivalent of unseen data from the real world for the model. This is to ensure that the model does not overfit on the validation set and provide high accuracies only there, but perform poorly on unseen data.

## CHAPTER-11

### RESULTS AND DISCUSSIONS

Many number of models were implemented and their accuracies were all compared against each other. Along with the models, different formats of data were also experimented with in terms of the features used.

Data Types :

1. Current bar notes : This data type uses each note in the bar for which the chord has to be predicted as features. For models where the data needs to be in a uniform shape, the bar is padded with -1 to make all bars of the same dimensions. This is an ideal case scenario and is impractical to be implemented as part of the model. This is because by the time all the notes for the current bar are played in real-time, it is time to play the chord for the next bar and not the current. The constraint of the real-time architecture rules out this type of data format in its purest form. A hybrid can be implemented which treats the previous bar as the current bar, and then uses some logic to predict what the following chord would be given the prediction for the previous bar.
2. Previous Bars : Here, a specified number of previous bars are taken into consideration and the first n notes of those bars are concatenated to form the features for prediction. The number of previous bars to take into account, and the number of notes to take from each of those bars are both parameterized within the yaml configuration file.
3. Previous Chords : In addition to the previous bars, the chord labels accompanying these bars can also be added to the input data under consideration. In many cases, like when using MultiLayer Perceptron with logistic and relu activations, adding the previous chords in addition to the previous bar data, improves accuracy by upto 10 percent.

4. Image Representation : This format is specific to only the CNN model. A convolutional neural network is primarily used for image classification, and the images are shaped as 2D matrices. A bar of music can also be represented as a matrix, where the rows represent each note of an octave and the columns represent time slices of the bar. A '1' within a cell represents the note for that row being played in the time slice for that column. Essentially, the bar is transformed into a matrix of 1s and 0s which is equivalent to an image.

Brief Introduction of the Models :

1. SVM : A Support Vector Machine attempts to find the “optimal” separating hyperplane for a dataset. Different kernels can be used for datasets that are not linearly separable. In the context of this project, each point is a bar of music with each note being a dimension. Each note of the previous bars can also be treated as a dimension.
2. MLP : A Multi Layer Perceptron is a fully connected neural network which can learn the decision surface of any function. The inputs in this case are the notes (represented as numbers) in both the current and previous bars format. The output of the network is a softmax probability indicating which of the chord labels is most probable.
3. LSTM : A Long Short Term Memory is an enhancement of a Recurrent Neural Network which has a concept of memory. The network knows what to remember and what to forget, and modifies the input across timesteps. Each timestep in this project, is a new note that was played. The model can work with both the current bar and previous bars data types as it can accommodate variable input lengths.
4. CNN : A Convolutional Neural Network uses learned filters to scan through an image and identify important features which are fed to a simple neural network which is then able to classify the image. A bar of music is transformed into an image-like matrix and these bars can then be classified thus predicting a label for the bar.
5. HMM : A Hidden Markov Model can be used to predict the most probable state sequence given an observation sequence provided the hidden states influence the

probabilities of observing the manifestations. In the context of this project, a chord is a hidden state and the notes are the manifestations. The model uses the previous bars data type, and the Viterbi algorithm to predict the most likely chord sequence given the notes sequence.

6. KNN-OMM Hybrid : A K-Nearest Neighbors model attempts to classify a label by looking at the closest neighbors of the query instance and taking a majority vote of the neighbors' classifications. An Observable Markov Model is a simple Markov Chain which learns probabilities based on sequences provided. This hybrid model uses KNN with the current notes of the previous bar as features to obtain a chord label for the previous bar. Using that classification, the OMM predicts the next chord in the sequence and that is used as the prediction for the current bar. This model, though naive, provides about ~35% accuracy on the test dataset.

Results :

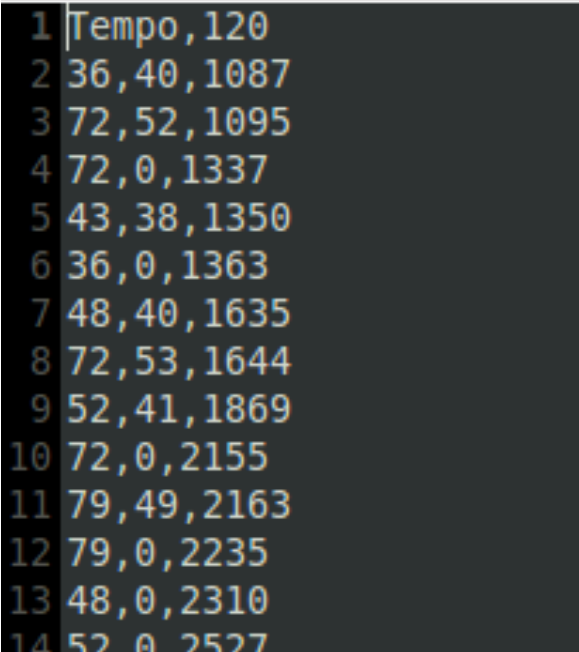
MODEL	DATA_TYPE	NOTES	ACTIVATION/KERNEL	CHORDS_IN_NRGAM	SCORES
SVM	current_bar	5	sigmoid	False	{'train': '0.242', 'test': '0.215'}
MLP	ngram_notes	10	identity	False	{'train': '0.319', 'test': '0.251'}
LogReg	ngram_notes	10	identity	False	{'train': '0.322', 'test': '0.255'}
LogReg	ngram_notes	10	relu	False	{'train': '0.322', 'test': '0.255'}
LogReg	ngram_notes	10	logistic	False	{'train': '0.322', 'test': '0.255'}
LogReg	ngram_notes	10	tanh	False	{'train': '0.322', 'test': '0.255'}
MLP	ngram_notes	10	tanh	False	{'train': '0.327', 'test': '0.271'}
LogReg	current_bar	5	identity	False	{'train': '0.324', 'test': '0.273'}
LogReg	current_bar	5	relu	False	{'train': '0.324', 'test': '0.273'}
LogReg	current_bar	5	logistic	False	{'train': '0.324', 'test': '0.273'}
LogReg	current_bar	5	tanh	False	{'train': '0.324', 'test': '0.273'}
MLP	ngram_notes	10	relu	False	{'train': '0.330', 'test': '0.283'}
MLP	ngram_notes	10	logistic	False	{'train': '0.308', 'test': '0.287'}
SVM	ngram_notes	10	rbf	False	{'train': '0.308', 'test': '0.287'}
SVM	ngram_notes	10	linear	False	{'train': '0.308', 'test': '0.287'}
SVM	ngram_notes	10	rbf	True	{'train': '0.308', 'test': '0.287'}
KO	None	15	None	False	{'train': '0.332', 'test': '0.291'}
SVM	ngram_notes	10	sigmoid	False	{'train': '0.279', 'test': '0.291'}
SVM	current_bar	5	rbf	False	{'train': '0.329', 'test': '0.302'}
SVM	current_bar	5	linear	False	{'train': '0.336', 'test': '0.302'}
SVM	ngram_notes	10	sigmoid	True	{'train': '0.302', 'test': '0.304'}
MLP	current_bar	5	tanh	False	{'train': '0.370', 'test': '0.305'}
MLP	current_bar	5	identity	False	{'train': '0.330', 'test': '0.309'}
MLP	current_bar	5	relu	False	{'train': '0.378', 'test': '0.309'}
MLP	current_bar	5	logistic	False	{'train': '0.344', 'test': '0.320'}
MLP	ngram_notes	10	logistic	True	{'train': '0.345', 'test': '0.344'}
PyHMM	sequence	1	None	False	{'train': '0.368', 'test': '0.348'}
PyHMM	ngram	1	None	False	{'train': '0.364', 'test': '0.352'}
MLP	ngram_notes	10	identity	True	{'train': '0.395', 'test': '0.352'}
SVM	ngram_notes	10	linear	True	{'train': '0.416', 'test': '0.360'}
LogReg	ngram_notes	10	identity	True	{'train': '0.409', 'test': '0.360'}
LogReg	ngram_notes	10	relu	True	{'train': '0.409', 'test': '0.360'}
LogReg	ngram_notes	10	logistic	True	{'train': '0.409', 'test': '0.360'}
LogReg	ngram_notes	10	tanh	True	{'train': '0.409', 'test': '0.360'}
MLP	ngram_notes	10	tanh	True	{'train': '0.447', 'test': '0.397'}
MLP	ngram_notes	10	relu	True	{'train': '0.464', 'test': '0.401'}

Fig. 11.1 Results

## CHAPTER-12

### SNAPSHOTS

Raw Data : First line contains the tempo, and every line after has note, intensity and timestamp (ms)



```
1 | Tempo, 120
2 | 36, 40, 1087
3 | 72, 52, 1095
4 | 72, 0, 1337
5 | 43, 38, 1350
6 | 36, 0, 1363
7 | 48, 40, 1635
8 | 72, 53, 1644
9 | 52, 41, 1869
10 | 72, 0, 2155
11 | 79, 49, 2163
12 | 79, 0, 2235
13 | 48, 0, 2310
14 | 52, 0, 2527
```

**Fig. 12.1 Raw Data**

Processed data which splits the left and right hand notes as well as calculates and arranges the data as per bars. Each line after the first is a bar. Left and right hand notes are separated by a comma. Each note is separated by a hyphen and note information is pipe separated.

```

1 Right Hand Notes, Left Hand Notes, Tempo=120, BarLength=2000.0
2 72|52|1095-72|0|1337-72|53|1644,36|40|1087-43|38|1350-36|0|1363-48|40|1635-52|41|1869
3 72|0|2155-79|49|2163-79|0|2235-79|51|2651-79|0|3077-81|47|3127-72|46|3130-72|0|3258-81|0|3430-81|48|3636,48|0|2310-52|0|2527-43|0|
2706-41|33|3121-53|35|3138-53|0|3365-48|33|3401-41|0|3403-48|0|3630-53|33|3648-53|0|3705
4 81|0|4138-79|50|4141-79|0|5136-77|53|5146-77|0|5399-77|50|5665,36|51|4171-43|38|4392-36|0|4494-48|45|4683-43|0|4737-48|0|4772-41|
5543-53|36|5667-48|0|5697-53|0|5823
5 76|46|6155-77|0|6175-76|0|6353-76|50|6645-76|0|7113-74|51|7117-74|0|7399-74|51|7633,48|36|6141-55|36|6619-48|0|6642-55|0|6684-43|
7425-55|42|7621-50|0|7675-55|0|7721
6 74|0|8127-72|41|8130-72|0|8814-81|35|9113-81|0|9207-79|50|9560,48|36|8126-55|29|8418-55|0|8715-48|0|8754-36|43|9573-43|41|9792-36|
7 77|57|10078-79|0|10086-77|0|10391-77|50|10569-77|0|11091-76|47|11102-76|0|11352-76|46|11605,41|40|10063-48|35|10329-41|0|10446-53|
10639-36|21|11101-43|36|11331-36|0|11559-48|36|11579-43|0|11622-48|0|11649
8 74|41|12066-76|0|12066-74|0|13428,
9 79|53|14072-79|0|14327-79|55|14613-79|0|15085-77|55|15109-77|0|15346-77|59|15622,36|49|14077-43|44|14320-36|0|14371-43|0|14587-48|
15103-48|41|15337-41|0|15397-53|50|15610-48|0|15630-53|0|15691
10 77|0|16113-76|55|16119-76|0|16343-76|58|16626-74|49|17140-76|0|17157-74|0|17618,48|44|16118-36|41|16121-43|29|16345-48|0|16348-36|

```

**Fig. 12.2 Processed Data - I**

Processed Chords data matches the left hand notes against an available chords dataset to find the match and adds the last column as the chord label. This is the final file in the data processing pipeline and is used for all models.

```

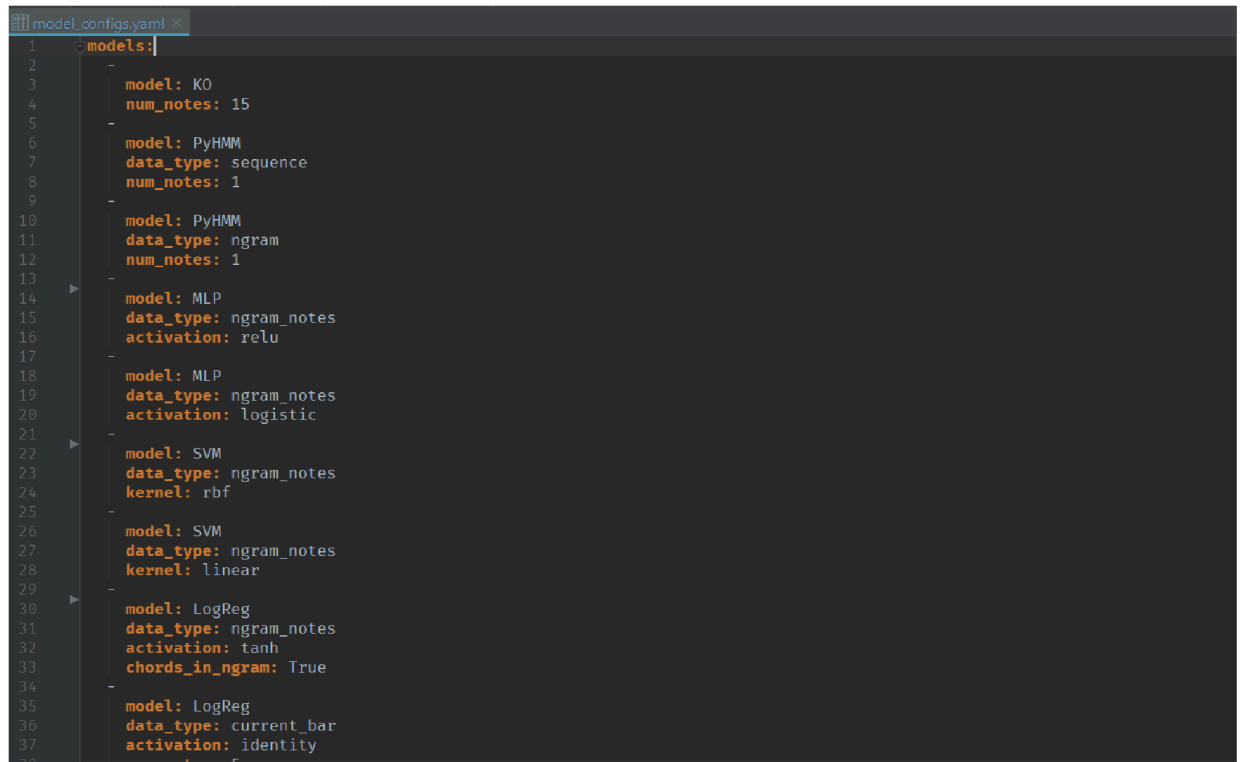
1 72|52|1095-72|0|1337-72|53|1644,36|40|1087-43|38|1350-36|0|1363-48|40|1635-52|41|1869,C M
2 72|0|2155-79|49|2163-79|0|2235-79|51|2651-79|0|3077-81|47|3127-72|46|3130-72|0|3258-81|0|3430-81|48|3636,48|0|2310-52|0|2527-43|0|
2706-41|33|3121-53|35|3138-53|0|3365-48|33|3401-41|0|3403-48|0|3630-53|33|3648-53|0|3705,C M
3 81|0|4138-79|50|4141-79|0|5136-77|53|5146-77|0|5399-77|50|5665,36|51|4171-43|38|4392-36|0|4494-48|45|4683-43|0|4737-48|0|4772-41|
5543-53|36|5667-48|0|5697-53|0|5823,F M
4 76|46|6155-77|0|6175-76|0|6353-76|50|6645-76|0|7113-74|51|7117-74|0|7399-74|51|7633,48|36|6141-55|36|6619-48|0|6642-55|0|6684-43|
7425-55|42|7621-50|0|7675-55|0|7721,G M
5 74|0|8127-72|41|8130-72|0|8814-81|35|9113-81|0|9207-79|50|9560,48|36|8126-55|29|8418-55|0|8715-48|0|8754-36|43|9573-43|41|9792-36|
6 77|57|10078-79|0|10086-77|0|10391-77|50|10569-77|0|11091-76|47|11102-76|0|11352-76|46|11605,41|40|10063-48|35|10329-41|0|10446-53|
10639-36|21|11101-43|36|11331-36|0|11559-48|36|11579-43|0|11622-48|0|11649,F M
7 79|53|14072-79|0|14327-79|55|14613-79|0|15085-77|55|15109-77|0|15346-77|59|15622,36|49|14077-43|44|14320-36|0|14371-43|0|14587-48|
15103-48|41|15337-41|0|15397-53|50|15610-48|0|15630-53|0|15691

```

**Fig. 12.3 Processed Data - II**

The snapshots describe the transformation of one song over the preprocessing pipeline. A whole folder full of such songs gets transformed and placed in different folders, eventually creating the complete dataset.

The below image shows a sample of the configuration of different models used for learning the mappings between bars and chords and predicting chord labels.



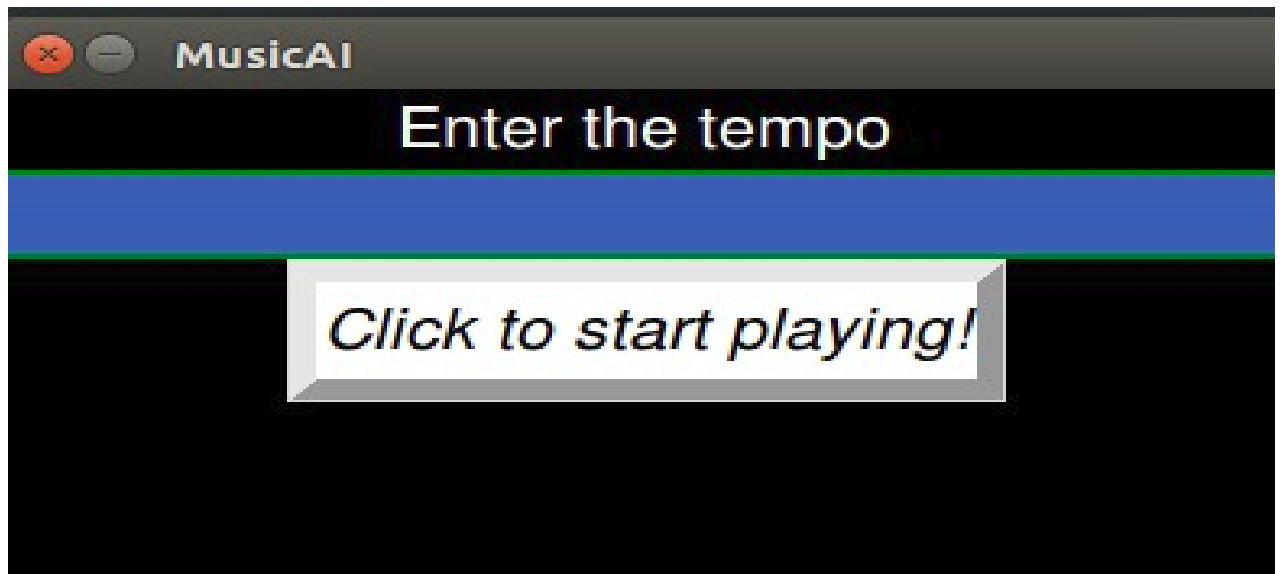
```

1 models:
2
3   model: KO
4   num_notes: 15
5
6   model: PyHMM
7   data_type: sequence
8   num_notes: 1
9
10  model: PyHMM
11  data_type: ngram
12  num_notes: 1
13
14  model: MLP
15  data_type: ngram_notes
16  activation: relu
17
18  model: MLP
19  data_type: ngram_notes
20  activation: logistic
21
22  model: SVM
23  data_type: ngram_notes
24  kernel: rbf
25
26  model: SVM
27  data_type: ngram_notes
28  kernel: linear
29
30  model: LogReg
31  data_type: ngram_notes
32  activation: tanh
33  chords_in_ngram: True
34
35  model: LogReg
36  data_type: current_bar
37  activation: identity

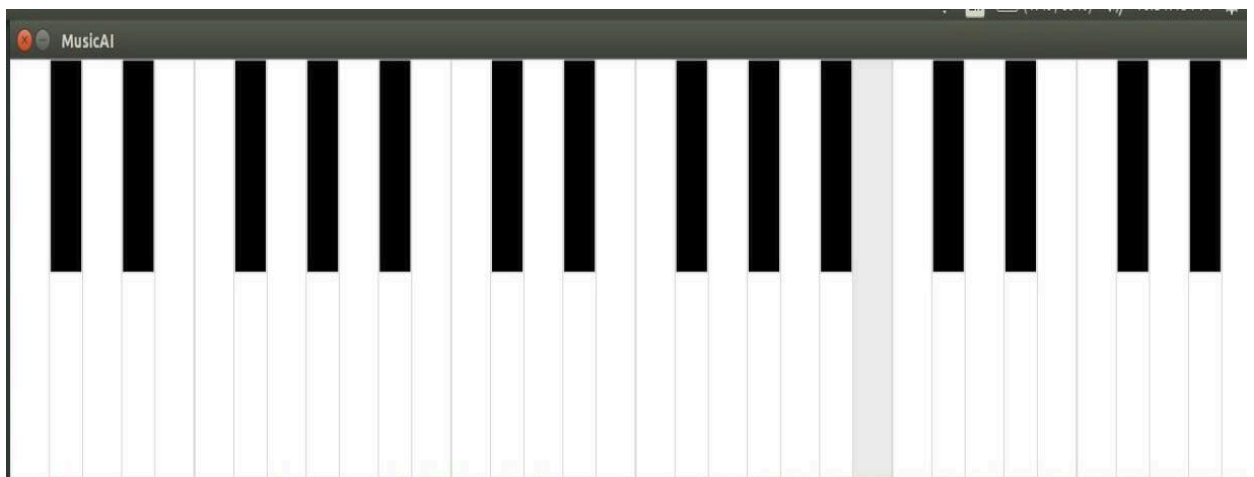
```

**Fig. 12.4 YAML config**

The entry point to this application is the GUI screen which prompts the user to enter the tempo, so that it could be communicated to the input as well as the output modules. The screen is shown below, where the entry box is focused on start of the application.

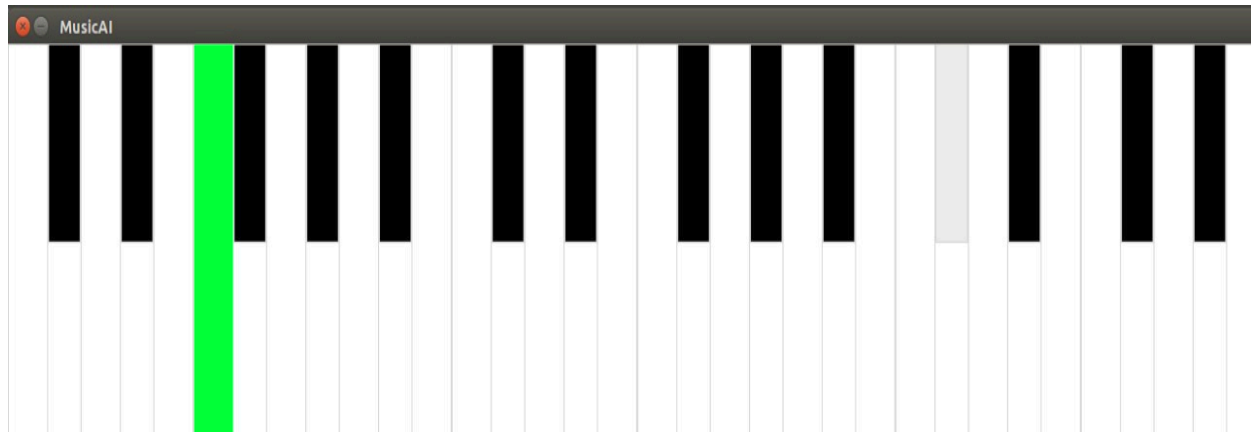
**Fig. 12.5 GUI - 1**

After the tempo has been entered, the core algorithm starts, which is described in the pseudo code section. Here, a keyboard layout is displayed to the user where the keys will light up if they are predicted by our model. The main aim in such an interface for the user is to view our keyboard and map it to his keyboard and play similar chords while playing again. The layout being displayed to the user is shown below.

**Fig. 12.6 GUI - 2**



The below image shows the state of the keyboard when the note was played. This is an indication to the user that a note is played, and he can learn a chord which needs to be played with the melody that he is playing.



**Fig. 12.7 GUI - 3**

## CHAPTER-13

### CONCLUSIONS

The current best performing model is the MLP classifier using a logistic activation function and lbfgs solver. It has an accuracy score of 48% on the test set.

The dataset is an inherently challenging one to learn and our results have set the baseline for future improvements.

An experiment was performed that shed light on the importance of accuracy in chord prediction for a model of this kind. A layman (not musically inclined) was asked to listen to two clippings of the same song. One clipping where the same chord was being played throughout (and thus resulting in low accuracy) and another where the chords played were as per the song's sheet music (high accuracy). It was observed that in most cases, the subject could not identify the difference between the two and would eventually resort to guessing which of the two was with high accuracy.

This result goes to show that music is somewhat arbitrary and a low accuracy of a mathematical model does not imply that the resulting music will sound bad. However, professional musicians will be able to identify the errors since they have a ear for it. Since the goal of the project is not to teach a beginner but to motivate him/her to keep playing/learning, this pitfall of a sub-par accuracy is acceptable to an extent.

## CHAPTER-14

### FURTHER ENHANCEMENTS

Further enhancements may include refining the models to increase accuracy of the prediction. However, it is empirically proved that higher accuracy does not imply better sounding music. A model with pure accuracy is also likely to sound sweet to a layman who does not have a keen ear for music.

The musical arpeggio patterns with which the model renders the left hand notes can be made more complex in order to make the model sound better. These patterns can be learnt as our existing dataset consists of a professional pianist's left hand notes played over the course of many songs.

The model can be made to play along with the user, regardless of the tempo. In other words, the model should estimate the speed at which the user is playing and play along appropriately. Currently, the user is expected to specify beforehand what the tempo of the song is.

A more cleaner and effective User Experience can be built by putting more thought into the GUI. As this project was only intended to be a prototype and not a full-fledged product, the GUI has been made basic.

## CHAPTER-15

### REFERENCES

1. Grosche, Peter, Meinard Müller, and Frank Kurth. "Cyclic tempogram—a mid-level tempo representation for musicsignals." *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on*. IEEE, 2010.
2. Chen, Ziheng, Jie Qi, and Yifei Zhou. "Machine Learning in Automatic Music Chords Generation."
3. Cunha, Uraquitan Sidney, and Geber Ramalho. "An intelligent hybrid model for chord prediction." *Organised Sound* 4.2 (1999): 115-119.
4. Gang, Dan, Daniel J. Lehmann, and Naftali Wagner. "Tuning a Neural Network for Harmonizing Melodies in Real-Time." *ICMC*. 1998.
5. Davies, Matthew EP, and Mark D. Plumbley. "Context-dependent beat tracking of musical audio." *IEEE Transactions on Audio, Speech, and Language Processing* 15.3 (2007): 1009-1020.
6. Leão, Henrique BS, et al. "Benchmarking Wave-to-MIDI Transcription Tools."
7. Buzzanca, Giuseppe. "A supervised learning approach to musical style recognition." *Music and artificial intelligence. Additional proceedings of the second international conference, ICMAI*. Vol. 2002. 2002.
8. Walker, J. "Midi-csv." URL: [http://www.fourmilab.ch/webtools/midicsv/\(hämtad 2017-05-20\)](http://www.fourmilab.ch/webtools/midicsv/(hämtad%2017-05-20)) (2008).
9. Anadon et al. "https://github.com/albertbou92/Automatic-Piano-Music-Generation"
10. Boulanger-Lewandowski, Nicolas, Yoshua Bengio, and Pascal Vincent. "Audio Chord Recognition with Recurrent Neural Networks." *ISMIR*. 2013.
11. Sheh, Alexander, and Daniel PW Ellis. "Chord segmentation and recognition using EM-trained hidden Markov models." (2003).
12. Bozhanov, Bozhidar. "Computoser-rule-based, probability-driven algorithmic music composition." *arXiv preprint arXiv:1412.3079* (2014).

13. Thom, Belinda, and R. Dannenberg. "Predicting chordal transitions in jazz: the good, the bad, and the ugly." *IJCAI-95, Music and AI Workshop Paper*. 1995.
14. D. P. Radicioni and R. Esposito. Advances in Music Information Retrieval, chapter BREVE: an HMPerceptron-Based Chord Recognition System. Studies in Computational Intelligence, Zbigniew W. Ras and Alicja Wieczorkowska (Editors), Springer, 2010.
15. Brossier, Paul M. "The aubio library at mirex 2006." *Synthesis*(2006).
16. Chuan, Ching-Hua, and Elaine Chew. "A hybrid system for automatic generation of style-specific accompaniment." *Proceedings of the 4th International Joint Workshop on Computational Creativity*. 2007.
17. Cheng, Heng-Tze, et al. "Automatic chord recognition for music classification and retrieval." *Multimedia and Expo, 2008 IEEE International Conference on*. IEEE, 2008.
18. Liang, Feynman. *BachBot: Automatic composition in the style of Bach chorales*. Diss. Masters thesis, University of Cambridge, 2016.
19. Eck, Douglas, and Jasmin Lapalme. "Learning musical structure directly from sequences of music." *University of Montreal, Department of Computer Science, CP 6128* (2008).
20. Tigas, Panagiotis. "Real-time jam-session support system." *arXiv preprint arXiv:1201.6251* (2012).

## CHAPTER-16

### BIBLIOGRAPHY

- Google AI Duet. <https://github.com/googlecreativelab/aiexperiments-ai-duet>
- BBC Vamp Plugins. <https://github.com/bbc/bbc-vamp-plugins/blob/master/README.md>
- BeatRoot. <https://code.soundsoftware.ac.uk/projects/beatroot-vamp>
- MARSYAS Feature Extractor. <http://marsyas.info/downloads/vamp-plugins.html>
- MELODIA - Melody Extraction. <http://mtg.upf.edu/technologies/melodia>
- Queen Mary plugin set. <https://vamp-plugins.org/plugin-doc/qm-vamp-plugins.html>
- Skytopia MP3 to MIDI. <http://www.skytopia.com/project/articles/mp3-to-midi.html>
- Mp3tomidi. <http://www.intelliscore.net/>
- WavToMidi. <http://www.pluto.dti.ne.jp/~araki/amazingmidi/>