American University of Armenia

Zaven & Sonia Akian College of Science and Engineering

CS251 Machine Learning Course Final Project

A machine learning approach to the classical music harmonization problem

Team:
Edgar Davtyan: edgar_davtyan2@edu.aua.am
Henrik Abgaryan: henrik_abgaryan2@edu.aua.am
Maro Zohrabyan: maro_zohrabyan@edu.aua.am
Aram Khanlari: aram_khanlari@edu.aua.am
Instructor:
Michael Poghosyan, Ph.D.: mpoghosyan@aua.am

Spring, 2021

Abstract

Melody is the basis for every musical piece. However, a world with only single-melody pieces would be a sad thing to imagine. Classical music through its existence has developed methods for accompanying the melody. One of the common tools is harmonization, which is the process of designating chords to each note of the melody. Chords typically consist of three notes.

Our project will discuss the problem of harmonizing a melody: assigning appropriate chords for a given melody or musical phrase. We will feed the machine learning algorithm pieces of classical music after a thorough analysis of which it will be able to deduce the principles of classical harmony and ultimately harmonize a given melody.

Keywords: Machine Learning, melody, harmony, supervised, unsupervised, harmonize, chord, classification, neural network.

Contents

Chapter 1

Introduction

1.1 A brief history of Music

Where words fail, music speaks.
-Hans Christian Andersen

Music is an indivisible part of our daily lives. Throughout history humanity's perception of music has evolved drastically. Having its' roots in simple rhythms produced when Neanderthals hit two pieces of stones together and branching to over 1300 genres, music owes a huge portion of its' development to Western European classical music.

Classical music's historical development has gone through many eras and to this day, musicologists and music historians work towards classification of classical music. One of the pillars of classical music, however, is melody. Melody can be defined as a linear succession of notes that the listener perceives as a single entity. In simpler terms, melodies are musical sentences. Melody's forms and its' definitions have varied a lot throughout history, but the fact that melody is the basis and foundation of every musical piece is an inarguable.

Despite being the building block of music, melody has changed its role through different eras and style of music. The journey begins with Gregorian chants and sequences, which are hymns performed in medieval Church tradition. Melodies here are simple and often with stable rhythms. Church music in this era (9-10th centuries) is *monophonic*, meaning that no matter how many people sing it, they always sing the same single melody.

Later on in development of music we witness the concept of Organa (plural of organum), which are earliest examples of *polyphonic* music, meaning that two or more melodies are combined at the same time. Note, however, that these melodies, despite sounding together are independent and equally important melodies. Polyphonic music in this was mostly performed by choirs, where different voices (groups of people) sing different melodies, or the same melody at different times. Centuries later, the concept was transmitted to instrumental music, reaching its climactic complexity in Johann Sebastian Bach's art.

Parallel to Bach's sophisticated polyphony, many composers such as Vivaldi and Handel created *homophonic* music. *Homophony* began by appearing in sacred music, replacing *polyphony* and *monophony* as the dominant form, but spread to secular music. *Homophony* is a musical texture in which a main melodic line is supported by one or more additional musical lines that add harmonic support. In fact, this is the musical texture that we hear most often today.

The *homophonic* tendencies continued and crystallized during the Classicist period. The Viennese classics, i.e. Joseph Haydn, Wolfgang Amadeus Mozart, and Ludwig van Beethoven composed in a style which established the rules for musical harmony. Undoubtedly, some of these rules were inherited from the previous musical traditions, but works of these three composers are the ultimate fundamentals for classical *homophonic* music.

Later in development of music, Romanticist composers largely used and developed the concept of harmony. In fact, this is the time period (19th century) when Harmony was formulated as a separate subject. Composers like Franz Schubert, Frederyk Chopin, Franz Liszt, Piotr Ilyich Tchaikovsky and Nikolai Rimsky-Korsakov have contributed to the development of harmony. The latter two have taught harmony and authored textbooks.

Future developments of harmony led to maximal enrichment of musical texture in music of Gustav Mahler, Sergei Rachmaninoff and Alexander Scriabin and eventually the collapse of harmony - atonal music in Arnold Schönberg's and contemporaries' art. Current composers have the luxury of returning to any of these time periods and composing in any harmonic style.

1.2 Understanding musical harmony

Having a general idea about the history of musical harmony let us now turn to its definition and key concepts. Harmony is often said to refer to the "vertical" aspect of music, as distinguished from melodic line, or the "horizontal" aspect. To clarify this definition let us picture melody as a function of time. So we have $M(t)$ which outputs certain note at a given time. Notes in music have many attributes such as dynamics (how soft/loud the note is), articulation (in an "angry", "sarcastic", "delicate", etc. manner), timbre (which instrument or voice performs it), etc., but in harmony we only consider their pitch (how low/high it sounds, which can be also determined by where it is located on the piano). For simplicity, let us take time intervals as integers, which will make $M(t)$ a discrete function. A simple example would be:

$$M(t), \text{ where } M(0) = 7, M(1) = 5, M(2) = 4, M(3) = 0$$

This corresponds to the first notes of the National Anthem of the Republic of Armenia. A small remark here: unfortunately, such mapping does not deal with the rhythm and repeating notes well. This will be discussed later

Having the function $M(t)$ (which will also be referred to as $M_S(t)$), the objective of harmony is to construct 3 different functions $M_T(t), M_A(t), M_B(t)$, which would harmonize i.e. support and serve as musical accompaniment for $M_S(t)$. The term "support" is vague and one of the reasons for that is that there are many possible harmonization combinations for a given melody $M_S(t)$. The choice of letters for the melody functions was not random. S stands for Soprano, A stands for Alto, T stands for Tenor and B stands for Bass. These are the four main types of human voices, which are present in the choir.

Let us restate once more. Harmony takes a melody $M_S(t)$ as an input and outputs 3 other melodies. As a result, these four melodies sound together forming consecutive *chords* $\Big\{M_S(i), M_T(i), M_A(i), M_B(i)\Big\}$ at time i and produce pleasant music.

Understanding whether the output is pleasant or unpleasant is a difficult, if not an impossible problem. Music, no matter how we compartmentalize it into mathematical terminology is not always measurable due to its highly personal impact. However, knowing theory of classical music and having many instances of classicist, romantic and (some) contemporary musical compositions one can take them as exemplary basis for constructing harmony.

1.3 Problem Setting and Project Motivation

Our project will discuss the problem of harmonizing a melody in classical style: assigning appropriate chords for a given melody or musical phrase. The motivation for such project had various sources.

Firstly, our group consists of one professional and one amateur musicians and two more members with great musical taste. Secondly, we truly wanted our project to give enjoyable results and music was one of the ways to achieve that. Thirdly, not initially, but during the process, our interest expanded to inclusion of not only classical music, but also folklore music into our trained models. This yielded interesting, but not fully explainable results.

1.4 Previous research on the problem

The problem of musical harmony has always been a matter of interest for computer scientists and recently machine learning specialists. Some of the interesting results in the field are the following.

A research tackling the problem of harmony generation using Deep learning and multi-objective evolutionary algorithm. [MT21] A project suggesting a solution for harmony in pop music. [Jun19] A project solving the problem with convolutional neural networks. [Gri19] A book, where harmony algorithm is proposed and applied to many other fields, such as solving NP-complete or Slope stability problems. [Gee09] A paper, proposing harmonic analysis tools, which is also helpful in understanding how to construct harmony.[RE07]

This, of course is not the comprehensive list of literature on the problem of harmony. Yet some of the presented papers have truly interesting results. However, they either solve a slightly different problem or suggest an advanced deep learning approach. Our project is a work in progress and includes a long list of future work, but has served well as an academic educational experience for all group members.

1.5 The Structure of the Project Paper

The rest of this report is organized as follows: in the next chapter we present the data collection and preprocessing techniques and performance measurements. In chapter 3 we discuss the initial algorithm we had projected and the models we have trained during the project. The fourth chapter is devoted to presentation of experiments and results. At the end we give a conclusion and a vision of further work. The paper includes an Appendix with the Python code, as well as bibliography used for literature research, coding and musical data in the end.

Chapter 2

Data and Preprocessing, Performance Measurement

2.1 Dataset

As stated before, our initial data was supposed to be pieces of classicist composers (Haydn, Mozart, Beethoven) whose music and classical harmony are largely codependent and basis for each other. After several attempts, we realized that in their original pieces, composers tend to improvise with the ways they serve harmony. A simple case of this can be found in Mozart's sonatas, where the harmony does not appear in the common "chord" form, but rather with consecutive notes or fast passages. Despite being an obstacle for us, this fact once again proved that musical genius is often unexplainable and surprising.

Our next step was to refer to choral pieces of these composers. Choir pieces usually preserve the style of chords we seek (four voices). But despite seeming like a good choice, this was also an unsuccessful attempt. Choirs were not famous among classicist composers. They either chose to compose for solo instruments, chamber ensembles or symphonic orchestras. The main problem in this case, was the small pool of pieces.

Our final attempt was the most logical choice but required an element of luck. As mentioned in the introduction, famous composers such as Piotr Ilyich Tchaikovksy and Nikolai Rimsky-Korsakov were also great lecturers and have written their own books on harmony. These textbooks include examples of solved problems (harmonized melodies) and examples from famous pieces - mostly in chord structure. They also offer many exercises to be harmonized by the students. Luckily we found online versions of some chapters of these books and multiple solved exercises from there. [RKAH05; Pis+71] On top of this, we came across a collection of solved harmony problems from a famous exercise book by Boris Alekseyev. We express our sincere gratitude to Sergei Schetnikov - the user who uploaded all the above-mentioned problems' solutions. [Mus]

A very small portion of the data was inputted by hand from the Armenian textbooks. [Pas87; Dub+78]

The process after the collection of data was interesting and challenging. All exercises were in different tonalities (had different "musical homes" that they start and finish at). One of the initial steps was to normalize the tonality. We have chosen C major and c minor tonalities for simplicity.

The next issue were the different time signatures (the concept of beats). This was solved after putting the data continuously after each other. Our main objective was to understand what chords are being used to harmonize melodies, so this issue was not prioritized. Some minor adjustments, such as removal of unnecessary text explanations, tempo markings and articulation also took place.

After the careful collection, normalization, and filtering of the data, we managed to produce 1165 bars of music. Each bar includes 1-8 different chords. The result is $\approx$ 39 minutes of continuous music. This data was sufficient for properly training our models and assessing the generated results. At first, we expected to have issues with the size of the file, but Google Colaboratory managed to handle the big data.

2.2 Feature selection and Preprocessing

When the Dataset with necessary amount of data was ready, the next obstacle was to interpret it in a language that would be understandable by our computer.

After thorough research, we have found a Python-based toolkit for computer-aided musicology provided by MIT called Music21. [21s] This library contains all the necessary functions to create, interpret, read and write music. To be able to work with our data the first thing was to convert *.midi* type file to something that we could work with easily.

We first introduced a function called **read-midi()** which is responsible for translating the music into computer language. The function takes the .midi file (note that there can be as many .midi files as you wish) as an input, first partitions the music according to the instrument using **partitionByInstrument()** function from Music21. After partitioning is over it starts to divide the music into 2 different groups: notes and chords. This is done using the respective Note and Chord classes. These notes and chords have string format and represent their real musical nature *(say for a note B in the* 3^{rd} *octave our program stores 'B3' and for C major chord in the* 4^{th} *octave it stores 'C4.E4.G4')*. In the end of the process, we have an array, where every entry is an array itself and contains already separated notes and chords. Now we have the format of the data, which can be modified easily if necessary.

The same function is later modified to convert the chords and notes into numbers in base 12 to be able to train some Classification and Regression models.

Traditionally, classical music has the notion of octave, which is mathematically speaking the doubling of Herzes. The note C say in the 3^{rd} octave has a frequency equal to 220 Hz, an octave distance from this note appears the note C in the 4^{th} octave with frequency $220Hz \cdot 2 = 440Hz$. This is applicable to all the notes in every octave. Thus the classical musical tradition has divided that octaves into 12 equal parts (called semitones). The initial note with the corresponding value of *0* is the note C. The meaning of every other number from 1 to 11 determines the semitone-distance of that note from the " origin.

With the above mentioned logic we brought all the notes and chords appearing in our melody to the base 12. Here it is crucial to understand that we do not take into account the order of octaves rather we are more interested in the notes used in the chords.

As previously mentioned we have combined our notes and chords into arrays. Another further modification that took place was removing from the array of chords single notes or chords consisting of 2 notes. The reasons behind this are rather theoretical. First of all our program is interested in 4 part harmony for 4 voices. We want to be able to find harmonization of such melodies which are accompanied with 3 other music lines i.e. previously mentioned $M_T(t), M_A(t)$ and $M_B(t)$. Secondly, there is no such definition of a chord consisting of 1 note. The chords consisting of 2 notes, on the other hand, are "meaningless" in terms of harmonization. That is if 2 notes appear in the chord it is impossible to say whether it has minor or major ("sad" or "happy") character and the harmonization will break, once trying to use this type of a chord.

After the data modifications we needed to transform it once more. To train classification models, Logistic Regression and Neural Network we need to construct feature-label sets. As the nature of our problem indicates, the predictions should be the chords that are going to harmonize our melody. That is why we stored

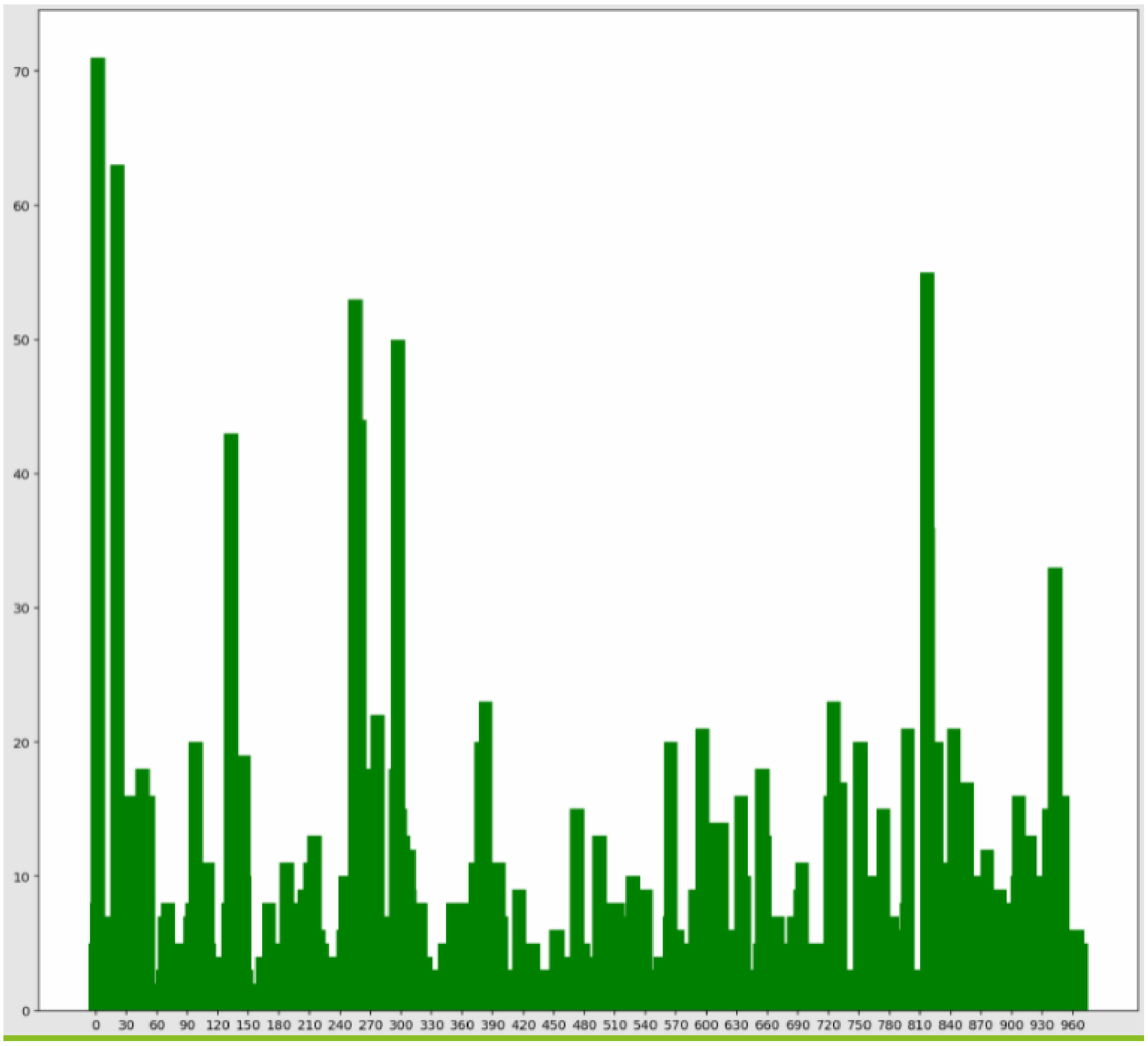
Figure 2.1: The distribution of chords according to the number of their occurrences

the notes as the features and the chords as their labels. Further, we used **one-hot encoding**, which represents an array of size 12 filled with 0's except for a 1 at the index of our feature (note). Recall that previously we have given unique values to each note (ranging from 0 to 11). To better understand the logic behind this let us construct the one-hot array for the features (notes) C and A. C, being the first note in our mapping has the value of 0, that means that in the one-hot array we will have 1 at index 0 and 0's at all other indices. Similarly, for A we will have a 1 at index 9 and 0's elsewhere:

C = [1,0,0,0,0,0,0,0,0,0,0,0]
A = [0,0,0,0,0,0,0,0,0,1,0,0]

2.3 Performance Measurement

In order to comprehend the performance of our models better we have calculated the mean of cross-validation scores for all the classification algorithms and plotted the respective boxplots to visualize the distribution.

As one can notice in the figure the classifications have almost equal means

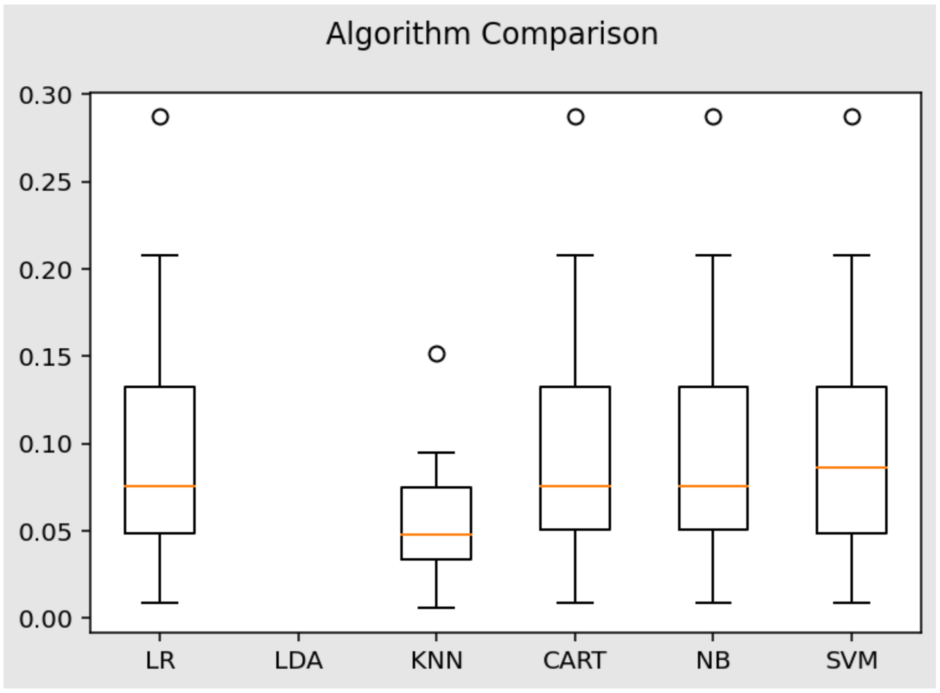

Figure 2.2: Boxplots for different models

and that the score is not that high. After changing the duration of the input data we were able to improve the performance of Logistic Regression going from initial 11% to 29%. One of the interesting cases we encountered was the fact that Linear Discriminant Analysis did not give any results in terms of cross-validation score. This is due to the fact that our data is not reducible in terms of features and it is not possible to draw linear boundaries for it. That is why the model could not give any correct prediction, resulting in *None* cross-validation score

We achieved this result after trying to train the model on both categorical *(this was the case when the notes and chords where in format of characters and strings: 'B3', 'C3.G5.B7')* and numerical *(here we used the base12 format of the same notes and chords)* feature-label pairs and finding out that as our data gets bigger the performance gets better.

The performance of the program was expected to be low in terms of accuracy score because of several facts:

1. for one feature in the train dataset it is very likely to have more than one label, which makes the probabilities of the appearing chords lower

2. in the case where the chords are not in base12 and retain their initial structure of strings, even the chords which consist of the same notes but are in different octaves are considered to be different. This is another reason which reduced the probabilities.

However, we need to make sure that in our project the accuracy score does not fully define the goodness of fit, because features can have multiple labels. The goodness of our program is measured by the number of perfectly harmonized notes. To make it clear, if the predicted chord "fits" well with the respective note in the melody we consider that to be a good fit.

Chapter 3

Algorithms and Models

3.1 Algorithms and Models

Our program trains several Classification and Regression models as well as Recurrent Neural Network on our data. Further we will go over all the algorithms that have been used and will discuss the details.

We came up with the idea of using Naive Bayes as the first try and further improve that algorithm considering the musical patterns. The following 2 algorithms explain the work in terms of music theory and the code provided in the Pseudocode appendix (later on: Pseudocode) is their implementation in Python.

1. Layer 1: Random
   The melody $A = \{x_1, x_2, \ldots, x_n\}$ is harmonized with random chords $C = \{c_1, c_2, \ldots, c_n\}$ where chords are chosen with no logic or distribution.

2. Layer 2: Pseudorandom
   The melody $A = \{x_1, x_2, \ldots, x_n\}$ is harmonized with chords $C = \{c_1, c_2, \ldots, c_n\}$ where a chord c_i is chosen such that it contains the note x_i. The decision for the chords is still random; however, the decision pool is narrowed down from all chords to those which contain the note

After training the model using these 2 algorithms and still getting not satisfactory results we have decided to train some Classification and Regression models such as:

- k-Neighbors Classifier,
- Support Vector Machine,
- Decision Trees,

- Random Forest, etc.

Here the performance was almost identical to Naive Bayes and we have decided to refer to Neural Networks.

The first try was to use the very basic type of Neural Networks or as it is more common to call it **Logistic Regression**. As mentioned in the Performance Measurement Section, we were able to boost the performance by manipulating with the data.

The other choice of model was the Neural Network. With the help of our professor Michael Poghosyan, Ph.D., we came upon **Recurrent Neural Networks**.

Recurrent neural networks (RNN) are networks with loops in them, allowing information to persist. [Ven19] They can be thought of as multiple copies of the same network, each passing a message to successor.

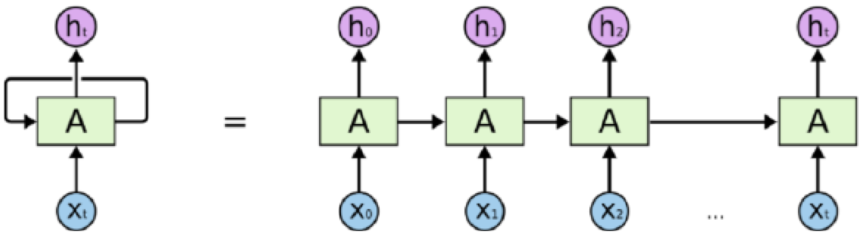

Figure 3.1: Recurrent Neural Network

Python has a library for RNNs, but we have decided to implement the network ourselves.

First we used the Keras library from Python in order to be able to use some activation functions, but as this library does not give a detailed explanation about the nature of the errors, we based our model on PyTorch.

As previously mentioned the neural network is based on PyTorch that is why the first step was to transform all the arrays in our program to torch.tensor-s. This was done to provide the smoothness of the program. We used the Softmax function as our activation function and other 256 hidden layers provided by the library. The learning rate was carefully chosen after number of trainings with different values and at the end arrived to a value of $5 \cdot 10^{-16}$. The same was done with iterations and the final value present in the code is 5000 *(note that we have tried to train the data with number of iterations equal to 10000, but the result was almost the same though it took longer to train)*.

The choice of the models was in some cases logical. For Naive Bayes and

Logistic Regression we wanted to estimate the probabilities of the chords and use that probabilities to see the goodness of the results. The Recurrent Neural Network on the other hand is proved to work well with complex data. In this particular case we needed the past information to be able to construct the harmony based on that.

This provides the smooth passages from one chord to another and makes the music pleasant to one's ear.

The other classification and regression models were tried to see the differences and patterns they follow. These models appear to give more or less good results compared with the previously mentioned ones. This program does not limit the choice of the model and it allows any other classification, regression or neural network algorithms to be trained on very different types of data.

Chapter 4

Results

4.1 Experiments and Results

The first experiments with our program started with exploring the Music21 library. This included splitting the Dataset into note-chord pairs, shuffling them, rearranging, etc. After we set up the first 2 algorithms we have trained our program on classical music only *(mostly Schubert and Chopin)*. Further improvements of our program allowed us to train it on different types of music starting with simple choral and reaching to traditional Yazidi. This was done to understand the strengths and weaknesses better and focus on improvements. Later we came across a better dataset which had all the necessary characteristics for our program to give as better results as possible.

To be able to identify the correctness of our harmonization we needed to convert our predictions to .midi files. The problem was solved when we came up with an idea of creating our own converter function.

The function **convert-to-midi()** takes the combination of the melody and predicted harmonization as an input. The function goes over the array, checks whether or not the element is an instance of Note or Chord, if it is a chord it splits it into notes considering the fact that the chords are in form of {*.*.*}. See the example in *Figure 4.1*.

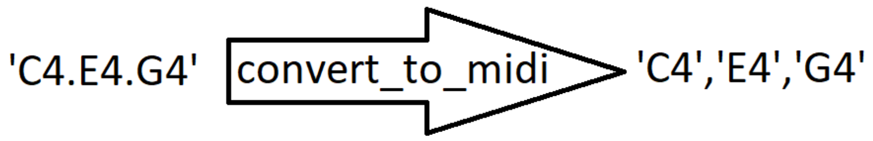

Figure 4.1: An example of a converted chord

With these characteristics it constructs a new array only consisting of notes. Later this final array is converted to .midi using the **Stream** class from Music21 library. The file is stored in the respective folder and it is possible to download and listen to it.

The function helped us to compare all the predictions during the whole process and note the improvements. From very low accuracy and a noise-like music we have reached something which is quite well harmonized and pleasant for the listener.

Chapter 5

Conclusion and Further work

5.1 Conclusion

Being the basis of musical piece, melody, however is not enough to express the whole range of emotions the composer feels. That is why classical music came upon some methods which would enrich the melody and make it more expressive. Harmonization is one of the most common from these methods and our aim was to create a program which would ultimately harmonize any given melody. The algorithms provided here work not only with classical but also with other types of music giving significant results. The harmonization of certain types of melody gives near 90% accurate results (in some of the cases the harmony was perfectly fit).

5.2 Further Work

Further work may expand to considering not only the standardized data (in our case the whole data is in 1 tonality) but also every other possible melody. Another improvement that can be done is to add the functionality of identifying the rhythm and duration of the notes.

There are some polyphonic rules in *Harmony* that are inherited from *Polyphony*. These rules are general patterns of acoustics. Some of such rules for example is if the base moves in one direction the melody is supposed to move to the opposite direction, direction meaning up and down. The later improvements of our program can detect whether or not the music follows those rules.

The concept of melodies which are not in a single tonality can be taken into consideration as well. Let us consider a melody that starts in C-major and then modulate to another tonality, for example A-minor. These two tonalities are called neighboring tonalities in musical terminology. This gives extra color and beauty to the music. As our problem solves the problem for only single tonality the further work can discuss the concepts of alternation and modulation.

Another modification can be to add intermediate notes which are means to move from one chord to another. These notes do not require separate chords for themselves but they enrich the existing chords and provide the smoothness of passages. The selection of such notes is not a straightforward task and this opens new horizons for re-harmonization and gives endless possibilities of a single melody to be harmonized.

Our program only considers the *Classical harmony*, but after the *Classical harmony* we have new set of more enriched rules in *Romantic harmony*. And then in *post-Romantic, Impressionist and Expressionist* music we have redefinitions and challenges of *Classical* and *Romantic harmonies*. These harmonic style are only a small portion of such in classical music. Further work may also include inclusion of jazz, pop and even techno harmonizations.

Lastly, we have to acknowledge the true value of this project serving as a first stair step in the beautiful dual reality of harmonic coexistence of computer science and music. Having the model for the harmonizing problem we may proceed to problems like orchestration, counterpoint creation in polyphony, fugue composition, harmonic analysis, musical style recognition, composing in certain style and even (might seem scary) composing based on big data of humanity's whole musical heritage.

5.3 Acknowledgements

We want to express our gratitude to prof. Michael Poghosyan, Ph.D. and Daniil Hayrapetyan for their support, encouragement and guidance throughout the whole process.

Chapter 6

Appendices

6.1 Pseudocodes

In this section we present pseudocodes of some of the important functions we use.

Algorithm 1: Arr To Eye

```
▷ Input: an array of integers $A$, each in the range 0-11
▷ Output: an array of one-hot arrays

1 Declare $result\_list = \emptyset$.
2 for $i \in A$ do
3     Declare an array $arr$ of size 12 predefined with zeros.
4     $arr[i] = 1$
5     append into $result\_list$ the array $arr$
6 return $result\_list$
```

Algorithm 2: Remove Double Single Chords

```
▷ Input: a list of chords $chords$
▷ Output: a list of chords with filtered out single and double sized chords

1 for $i \in chords$ do
2     if $i$ has $\leq 2$ notes then
3         remove $i$ from $chords$
4 return $chords$
```

Algorithm 3: Convert to Midi

```
▷ Input: A predicted composition $P$
▷ Output: A set of chord and note objects based on the composition

create an empty set of musical objects $output = \emptyset$.
for each $pattern \in P$ do
    if pattern is a chord then
        create an empty set of notes $notes = \emptyset$.
        for each note $n \in pattern$ do
            create a Note object $note = pattern$
            make sure note stores a piano sound
            add note into notes
        create a Chord object chord
        add into chord the set of notes notes
        add into output object chord
    else
        create a Note object $note = pattern$
        make sure note stores a piano sound
        add note into output
return output
```

Algorithm 4: Read Midi

```
▷ Input: A midi file $P$
▷ Output: A set of chord and note objects

create an empty set of chords $chords = \emptyset$.
create an empty set of Notes $notes = \emptyset$.
create a set of all instruments in the file $P$
for each $part \in P$ do
    for each $element \in part$ do
        make sure element stores piano sounds
        if element is a Chord then
            a
        dd element into chords
    else
        add element into notes
return notes, chords
```

For more details, please, see the actual code [MZ21].

Bibliography

[Pis+71] Walter Piston et al. "Guide to the Practical Study of Harmony". In: *Notes* 27.4 (1971), p. 707.

[Dub+78] I Dubovsky et al. *Harmony Textbook*. Luys, 1978.

[Pas87] Eduard Pashinyan. *Harmony*. Soviet Writer, 1987.

[RKAH05] Nikolay Rimsky-Korsakov, Joseph Achron, and Nicholas Hopkins. *Practical manual of harmony*. C. Fischer, 2005.

[RE07] Daniele Radicioni and Roberto Esposito. "Tonal Harmony Analysis: A Supervised Sequential Learning Approach". In: vol. 4733. Sept. 2007, pp. 638–649.

[Gee09] Zong Woo. Geem. *Music-Inspired Harmony Search Algorithm Theory and Applications*. Springer Berlin Heidelberg, 2009.

[Gri19] Luke Griswold. *Using Convolutional Neural Networks to Generate Harmony*. en. Dec. 2019.

[Jun19] Haebichan Jung. *Making Music: When Simple Probabilities Outperform Deep Learning*. en. Jan. 2019.

[Ven19] Mahendran Venkatachalam. *Recurrent Neural Networks*. 2019.

[MZ21] H. Abgaryan E. Davtyan M. Zohrabyan A. Khanlari. *Project Source Codes*. https://github.com/EDavtyan/ML-CS251-340A. 2021.

[MT21] Maryam Majidi and Rahil Mahdian Toroghi. "Music Harmony Generation, through Deep Learning and Using a Multi-Objective Evolutionary Algorithm". In: *CoRR* abs/2102.07960 (2021). arXiv: 2102.07960.

[Mus] Musescore. *Sergey Shchetnikov*.

[21s] *music21: A toolkit for computer-aided musicology*.