POLITECNICO DI MILANO
DIPARTIMENTO DI ELETTRONICA E INFORMAZIONE
CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

CHORD SEQUENCES: EVALUATING THE EFFECT OF COMPLEXITY ON PREFERENCE

Master graduation thesis by :
Francesco Foscarin

Supervisor:
Prof. Prof. Augusto Sarti
Assistant supervisor:
Prof. Dr. Bruno Di Giorgi

Academic Year 2016-2017

Abstract

WHAT is the level of complexity that we like most when we listen to music? This is the fundamental question that motivates this work. It is not easy to give an answer; yet, music recommendation systems would benefit enormously from an algorithm that could use a currently unused musical property, such as complexity, to predict which music the users prefer. In order to answer this question, we exploit the subdivision of music in multiple aspects, such as chords, rhythm, melody, orchestration, etc. and we proceed by using the "divide et conquer" strategy, addressing specifically the task of understanding the relation between preference and complexity of chord sequences.

We know from previous research that some properties, e.g. complexity, tend to increase the amount of activity in the listener's brain (a.k.a. arousal potential) and we know that this parameter influences the listener's preference for a certain piece of music. In particular, there is an optimal arousal potential that causes maximum liking, while a too low as well as a too high arousal potential results in a decrease of liking.

We hypothesize that the the level of the optimal arousal potential depends on the music expertise of a person. Therefore, we design a listening test in order to prove the existence of a relation between these two values. As a measure of complexity, we use a set of state of the art models of harmonic complexity, based on chord sequence probability. Our results, analyzed using robust regression techniques, show a positive correlation between the musical expertise of the subjects who participated in the test and the complexity of the chord sequences that they chose as the one they prefer.

Summary

QUAL è il livello di complessità che preferiamo durante l'ascolto di musica? Questa è la domanda fondamentale che motiva il nostro lavoro. Nonostante sia complicato trovare una risposta adeguata, i sistemi di raccomandazione in ambito musicale trarrebbero grandi benefici da un algoritmo in grado di usare una proprietà musicale attualmente inutilizzata (la complessità) per suggerire la musica preferita dagli utenti. Per rispondere a questa domanda, sfruttiamo la suddivisione della musica in molteplici aspetti, come gli accordi, il ritmo, la melodia, l'orchestrazione, ecc... e utilizziano la strategia "divide et impera", considerando solamente la relazione tra preferenza e complessità delle sequenze di accordi.

Sappiamo da precedenti ricerche che alcune proprietà, tra cui la complessità, tendono ad aumentare la quantità di attività cerebrale dell'ascoltatore (il cosidetto "arousal potential"). Inoltre sappiamo che questo parametro influenza la preferenza dell'ascoltatore. In particolare è possibile individuare un "arousal potential" ottimale che produce il massimo gradimento, mentre livelli troppo alti e troppo bassi di questo fattore determinano un minor apprezzamento del brano.

In questo lavoro ipotizziamo che il livello ottimale di "arousal potential" dipendenda dall'esperienza musicale di ogni individuo. Pertanto, progettiamo un test di ascolto che punta a verificare l'esistenza di una relazione tra questi due valori. Come misura della complessità usiamo un insieme di modelli di complessità allo stato dell'arte, che si basa sulla probabilità delle sequenze di accordi. Analizzando i nostri risultati attraverso tecniche di regressione robusta, proviamo l'esistenza di una correlazione positiva tra l'esperienza musicale del soggetto e la complessità delle sequenze di accordi da lui preferite.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

The goal of this thesis is to give an answer to the question: "What is the level of complexity that we like most when we listen to music?". As we will see, an answer is not easy to give, although the human intuition is able to classify two comparable objects in more and less complex, and more and less pleasing. In order to have a complete overview of this topic, we start by defining the meaning of complexity and, in particular, musical complexity. We then deal with the relation between complexity and pleasure that has been found by researchers and, finally, we discuss how this thesis will help give an answer to the question that we presented at the beginning.

1.1 Complexity

Complexity is a property that can be attributed to a vast set of stimuli. One of the earliest attempts to define it was made by Birkhoff in his book "Aesthetic Measures" [6], where he writes that complexity can measure the extent to which an object calls for an "effort of attention" and a "feeling of a tension". This definition well delineates what is the human intuition of complexity, but it provides little, if any, guidance toward an actual measure of it. Without doubt, the term complexity is understood differently in different contexts and lacks a clear and unified definition. Streich ironically grasps the difficulty of its definition by considering it an autological term, since in a way it can be applied to itself: the term complexity is complex [60]. He distinguishes between two different perspectives, that he calls the *formal view* and the *informal view*. The former is related to scientific fields, where a precise expression was necessary. In computa-

Figure 1.1: *Complete order (left), chaos (center) and complete disorder (right) [Figure taken from [18]].*

tional theory, complexity indicates the amount of resources required for the execution of algorithms; for example, the time complexity of a problem is equal to the number of steps that it takes to solve an instance of the problem using the most efficient algorithm. In information theory, instead, it is a measure of the total amount of information transmitted by a source with every symbol. In network theory complexity is the product of richness in the connections between the components of a system. The informal view is more interesting for us because it covers the everyday, non-scientific understanding of this property and therefore can be related directly to cognitive problems. We could say that something is considered complex not only if we have difficulties describing it, but even if we don't really understand it. However, this is only the case as long as we still have the impression or the feeling that there is something to understand. Opposed to this, if we do not see any possible sense in something and we have the feeling that it is completely random, we would not attribute it as complex [60]. A very clear example of this behavior is given by Edmonds [18] in an perception experiment with the three pictures of Figure [18]. While the figure representing order (left) was indicated as not complex and the central figure was indicated as complex, some people hesitated between the middle and right panels when being asked to point out the most complex one. But once told that the right one was created by means of a (pseudo-) random number generator, the right panel was usually no longer considered as complex.

1.1.1 Musical Complexity

To define a meaning of complexity for music we must first of all emphasize the concept of music as composed by an highly structured hierarchy of different layers. Taking as example a generic pop song, at the base of the hierarchy we find single frequencies, that can be grouped in a more structured way (pitches) or in a more chaotic way (noisy sounds). Every sound then, can have a different temporal envelope. Another higher layer is rhythm, i.e. the exact pattern of disposition of the sounds in the time axis. The pitches can then be grouped in chords and chords can be combined sequentially to form chord progressions. If we keep going up in the hierarchy, after many other layers such as chord voicings, arrangement, orchestration and lyrics, we will finally have the complete songs.

Complexity in music can be generated by each single layer of this hierarchy and by any combination of them. To address the complete musical complexity, it is therefore convenient to use the "divide and conquer" strategy, by working separately with different

layers. Fortunately, this division is a common practice of musical theory and musical analysis, so we have an already well developed framework to address each facet separately.

In this thesis we will deal with the complexity generated by sequences of chords (we provide a complete definition of this term in Section 2.1.5). Looking back at the experiment of Edmonds, we can predict that chord sequencees with a defined structure will sound not complex and that the complexity will increase while the sequences become more "chaotic". However, not every combination is possible, since progressions generated randomly, with no rules at all, will have a complexity level that is hard to compare and define. A translation in the musical field for the terms "defined structure" and "chaotic" will be proposed later.

1.2 Complexity and Preference

Let us reconsider our initial question about the correlation between the complexity level and the preference rating. It is a common knowledge that a too simple stimulus will result as banal and boring, but we also know that a too complex one will be perceived as strange and unpleasant. Therefore, a first approximation of the answer would be that we like a level of complexity that stands in between. Many psychologists tried to arrive to a more precise answer but they demonstrated that sometimes more and sometimes less complex stimulus patterns are more attractive [3]. This random behavior suggest that maybe the research of the optimal level of complexity cannot be performed, without taking into accounts the dependency from other factors. Some studies were made about the relation between complexity and novelty. Berlyne [2] repetitively proposes to a subject black and white reproductions paintings with different level of complexity: more complex paintings (crowded canvases with many human figures), less complex paintings (portraits of a single person), more complex nonrepresentational patterns (containing many elements), and less complex nonrepresentational patterns. He observes that, with repeated exposition, the less complex pictures were rated significantly more and more displeasing, while the ratings for the more complex pictures did not change significantly. Moving to the music field, similar results where achieved by Skaife in [59]. He shows that preference tends, when "simple" popular music is presented, to decline with increasing familiarity. But with repetitive hearing of jazz, classical, or contemporary music that violates traditional conventions, there is normally a rise followed by a decline. Similar results were reached by Alpert [1] with rhythmic patterns.

A summary of these and others related works was made by Berline in [3]. He used all these results to provide a confirmation of his theory that an individual's preference for a certain piece of music is related to the amount of activity it produces in the listener's brain, which he refers to as the *arousal potential*. In particular, there is an optimal arousal potential that causes the maximum liking, while a too low as well as a too high arousal potential result in a decrease of liking. He illustrates this behavior by an inverted U-shaped curve (Figure 1.2). Moreover he identifies as the most significant variables affecting the arousal property as complexity, novelty/familiarity and surprise.

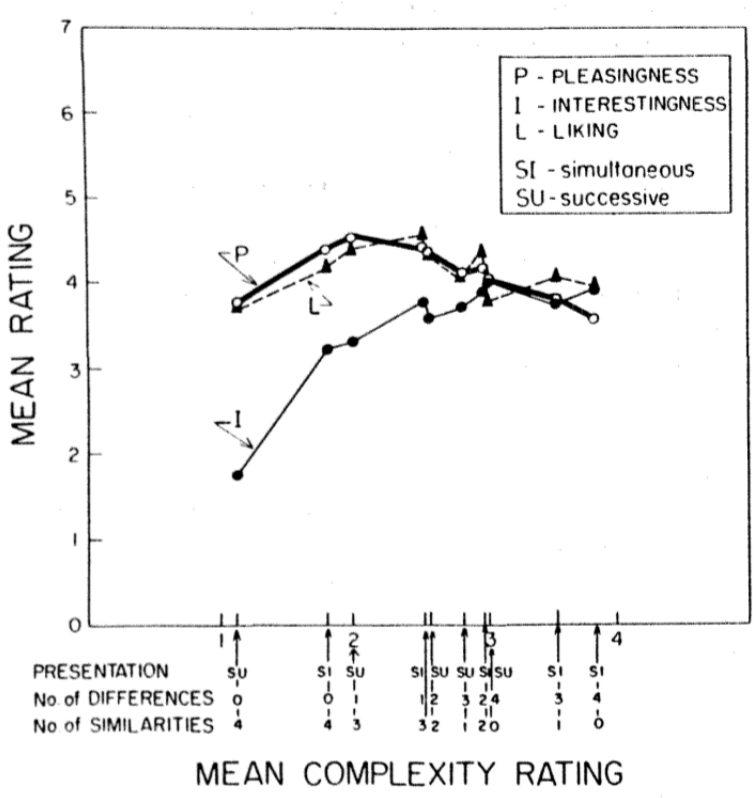

Figure 1.2: *The inverted U-shape curve generated by complexity and preference [Figure from [3]].*

1.3 Music Information Retrieval - MIR

Extracting the degree of complexity and the preference rating from a song is a task that belongs to the field of Music Information Retrieval (MIR). MIR is a branch of information retrieval that deals with the research of strategies for enabling access to digital music collections, both new and historical. On-line music portals like last.fm, iTunes, Pandora, Spotify and Amazon disclose millions of songs to millions of users around the world and thus the shift in the music industry from physical media formats towards online products and services is almost completed. This has put a big amount of interest in the development of methods to keep up with expectations of search and browse functionality. In a similar context the "old" approach of making experts manually annotate audio with metadata is not a feasible solution anymore, since the cost to prepare a database to contain all the information necessary to perform similarity-based search is enormous. It has been estimated that it would take approximately 50 person-years to enter the metadata for one million tracks [9]. On the contrary, an approach that could deal with the continuous growth of musical content is to automatically extract from low level information (audio or the symbolic representation of musical content) some high level features such as genre, melody, attitude of the listener (e.g. if a song sounds happy or depressing), etc.

As a low level information to start with, we will use a particular form of music symbolic representation: chord labels. They are a flexible and abstract representation of musical

harmony that developed with the emergence of jazz, improvised, and popular music. They can be considered as a rather informal map that guides the performers, instead of describing in high detail how a piece of music should be performed. However, also in music theory, music education, composition and harmony analysis, chord labels have proved to be a convenient way of abstracting from individual notes in a score [16].

It is clear that MIR systems would benefit enormously from an algorithm that could use a currently unused musical property (i.e. complexity), to suggest what music the user will prefer. In particular, that would improve music classification and recommendation systems.

1.4 A Measure of Preference for Chord Sequences Complexity

In order to classify the correlation between the preference and the complexity of the chord progressions, we need to find a reliable measure of their complexity.

For this task we will use the results obtained by Di Giorgi in [17]. He considered three different machine learning models: prediction by partial matching, Hidden Markov Model and Recurrent Neural Network. These models were then trained using a very large dataset containing the chords annotation of half a million songs, and a new model, a weighted combination of the three, was created in order to maximize the model performances over the dataset. With a listening test he showed the existence of a strong positive and statistically significant correlation between the log probability of the chord sequences, computed by the model, and their complexity ratings.

We will start by using the same models trained by Di Giorgi but we propose a different technique for the generation of chord sequences that will avoid them to become too "randomic" and not understandable. We also propose a different listening test, tuned to give the user the possibility of actively choosing the preferred chord sequence. The goal is to investigate if there is a correlation between the musical expertise of a person and the degree of chord sequences complexity that he or she prefers.

1.5 Thesis Outline

The remainder of this thesis is structured as follows.

In Chapter 2 we provide some basic concepts of music harmony and mathematics that are necessary for the understanding of the next chapters. Then, we briefly illustrate a series of works that are related to this thesis, such as the state of the art for preference estimation from chords progression complexity, and other works that inspired the methodology that we used for our experiment.

In Chapter 3 we present the machine learning models that we will use in our experiment focusing on their structure and explaining the different approach that they use in order to "learn" from data. We also provide a short description of the training techniques and the dataset used by Di Giorgi in [17].

In Chapter 4 we describe how we used the machine learning models in order to generate harmonic progressions. Then we deal with the design of the experiment, as well as the specific technology that we used to implement it efficiently. Finally we present the results of our experiment.

In Chapter 5 we summarize the main contributions of our work and we propose some future research that could follow from this work.

CHAPTER 2

Background

In this chapter we define the concepts necessary for the understanding of the discussions in the next chapters. We introduce in Section 2.1 some notation about Tonal Harmony, focusing on scale, interval and chord. In Section 2.2 we resume the basics of probability and machine learning. Afterwards, in Section 2.3 we explore the related works and we give an overview of the state of art for the computation of an harmonic complexity index.

2.1 Tonal Harmony

In music, harmony considers the process by which the composition of individual sounds, or superpositions of sounds, is analyzed by hearing. The word *Tonal*, instead refers to music where one sound (the tonic) has more importance than other sounds and this definition includes the majority of music produced in western culture from approximatively 1600. In this thesis we leave aside other traditions of music, since they are based on different concepts and would require a different approach.

2.1.1 Pitch

Pitch is a perceptual property of sounds that allows their ordering on a frequency-related scale [31].

Table 2.1: *The scientific pitch notation (SPN) for pitches, assigns a frequency to each pitch in a specific octave.*

	Octave 0	Octave 1	Octave 2	Octave 3	Octave 4	Octave 5	Octave 6	Octave 7	Octave 8
C	16.35	32.70	65.41	130.81	261.63	523.25	1046.50	2093.00	4186.01
C#	17.32	34.65	69.30	138.59	277.18	554.37	1108.73	2217.46	4434.92
D	18.35	36.71	73.42	146.83	293.66	587.33	1174.66	2349.32	4698.64
D#	19.45	38.89	77.78	155.56	311.13	622.25	1244.51	2489.02	4978.03
E	20.60	41.20	82.41	164.81	329.63	659.26	1318.51	2637.02	5274.04
F	21.83	43.65	87.31	174.61	349.23	698.46	1396.91	2793.83	5587.65
F#	23.12	46.25	92.50	185.00	369.99	739.99	1479.98	2959.96	5919.91
G	24.50	49.00	98.00	196.00	392.00	783.99	1567.98	3135.96	6271.93
G#	25.96	51.91	103.83	207.65	415.30	830.61	1661.22	3322.44	6644.88
A	27.50	55.00	110.00	220.00	440.00	880.00	1760.00	3520.00	7040.00
A#	29.14	58.27	116.54	233.08	466.16	932.33	1864.66	3729.31	7458.62
B	30.87	61.74	123.47	246.94	493.88	987.77	1975.53	3951.07	7902.13

An *octave* is the distance between a musical pitch and another with half or double its frequency. The octave relationship is important for our perception of music, because the human ear tends to hear two pitches at distance of one or more octaves, as being essentially the same pitch.

The equal temperament system divides each octave in 12 pitches, with the formula:

$$f_p = 2^{1/12} f_{p-1} \tag{2.1}$$

Pitches can be labeled using a combination of letters and numbers, as in scientific pitch notation, that assigns a frequency to each pitch in a specific octave (Table 2.1).

The terms pitch, tone and note are used as synonyms throughout this thesis.

2.1.2 Scale and Intervals

A sequence of ordered pitches is called a scale. Two scales are used as the basis of Western music: major scale and minor scale (with its harmonic and melodic forms) [53]. Another important scale is the chromatic scale, characterized by the fact that it is an ordered sequence of all the pitches. When the utilization of a particular scale is defined (or implicitly given from the context), it is common to refer to each step of the scale with the term *degree*. For example, in the major scale of Figure 2.1, the F note is called "fourth degree", since it is the forth note of the scale from the tonic. Scale degrees are often notated using roman numbers.

The basis of harmony is the *interval*. This name is used to describe the "distance" between the two tones, measured by their difference in pitch. If the two tones are not heard at the same time, but are consecutive tones of a melody, the interval is called a *melodic interval*, as distinguished from the *harmonic interval* in which the two tones are sounded together (Figure 2.2).

The smallest music interval is called *semitone* (a.k.a. half step or half tone) and it is defined as the interval between two adjacent notes in a chromatic scale. Two semitones form a *tone*. The bigger intervals can be defined using the ones founding the major scale

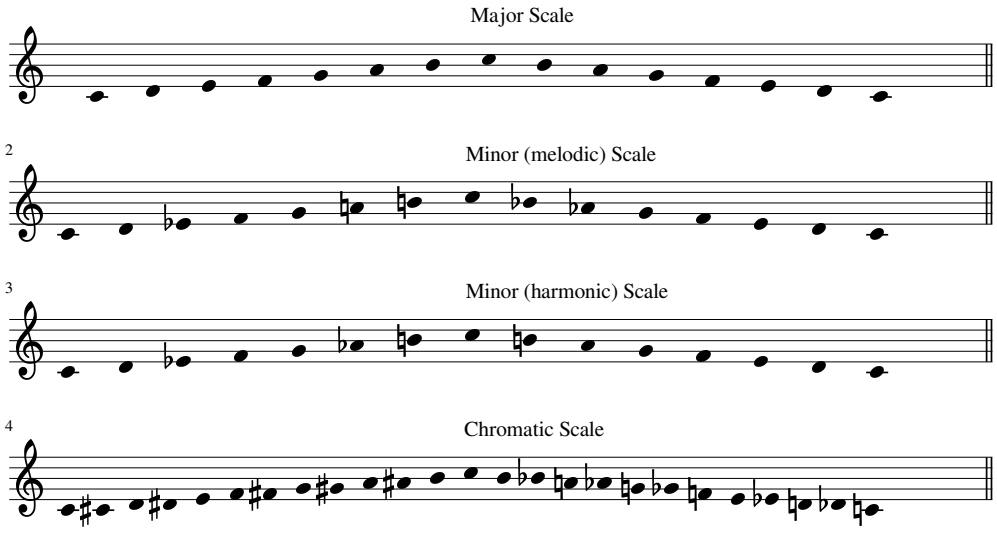

Figure 2.1: *The most used scales in Western music: major scale, minor scale (with its harmonic and melodic forms), and chromatic scale.*

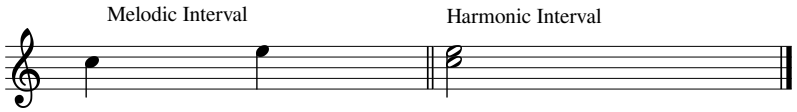

Figure 2.2: *Melodic Interval (consecutive tones) and Harmonic Interval (simultaneous tones).*

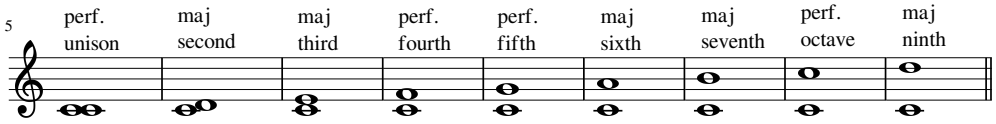

Figure 2.3: *The common names used for the intervals between the starting note of a major scale and the other notes.*

Table 2.2: *The most used interval labels in music theory with the corresponding number of semitones.*

Interval Name	Note Example	# of Semitones
unison	C - C	0
aug. unison	C - C#	1
min. 2nd	C - Db	1
maj. 2nd	C - D	2
aug. 2nd	C - D#	3
min. 3rd	C - Eb	3
maj. 3rd	C - E	4
aug. 3rd	C - E#	5
perf. 4th	C - F	5
aug. 4th	C - F#	6
dim. 5th	C - Gb	6
perf. 5th	C - G	7
aug. 5th	C - G#	8
min. 6th	C - Ab	8
maj. 6th	C - A	9
aug. 6th	C - A#	10
min. 7th	C - Bb	10
maj. 7th	C - B	11
octave	C - C	12

as a reference; they can be defined in term of semitones or tones, but it is common in music theory to refer to them with another notation (Figure 2.3). Among the intervals from the major scale, the second, the third, the sixth and the seventh are said *major*; Octave, fifth, fourth and unison are instead defined with the term *perfect*. If the upper tone does not coincide with a note of the scale, the following considerations are to be applied:

- An interval a half-tone smaller than a major interval is *minor*.
- An interval a half-tone larger than a major or perfect interval is *augmented*.
- An interval a half-tone smaller than a minor or a perfect interval is *diminished*.

From the Table 2.2 it is possible to see that it often happens that two intervals with a different name correspond to the same number of semitones. A good example is the augmented second, which cannot be distinguished from the minor third, without further evidence than the sound of the two tones. In this situation, one interval is called the *enharmonic equivalent* of the other. Explaining the reason why different names are necessary for the enharmonic equivalent intervals is beyond the purpose of this small introduction.

A common operation regarding intervals is the *harmonic inversion* or simply *inversion*.

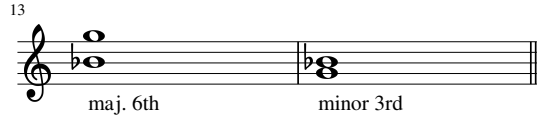

Figure 2.4: *The harmonic inversion of a major sixth. This operation can change the name of the interval.*

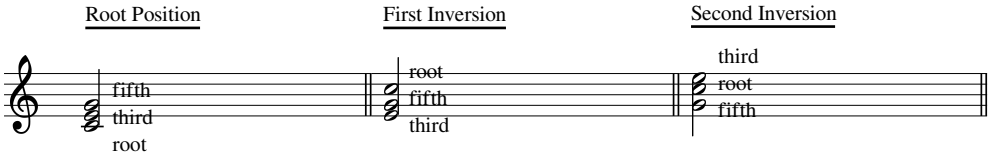

Figure 2.5: *The three possible inversions of a triads. It is possible to still uniquely identify the note generated by the superposition if third, using the names "root", "third" adn "fifth".*

In this procedure the names of the notes remain the same, but the lower of the two becomes the upper (or vice versa) with the consequence that there is usually a change in the name of the interval (Figure 2.4).

2.1.3 Triads

A *Chord* is the superposition of two or more intervals. The simplest chord is the *triad*, a chord of three tones obtained by the superposition of two thirds. The triad is the basis of the Western harmonic system. The names *root* (or *fundamental*), *third* and *fifth* are given to the three tones of the triad. These terms allow to identify the tones even if the notes can be duplicated at different octaves or rearranged in a different order (Figure 2.5).

A triad with its root as its lowest tone is said to be in root position. A triad with its third as its lowest tone is said to be in the first inversion. A triad with its fifth as its lowest tone is said to be in the second inversion. Changing the lowest note is an operation that can change the way it is perceived. There are other possible operations that can be performed on the notes of a triad; for example it is possible to move notes or duplicate them at different octaves, giving us a lot of possible ways of playing the same chord.The specific way the notes of a chord are assembled is called *voicing*.

Taking the scales, major and minor and using only the notes of these scales, superposition of third gives triads that differ depending on the kind of third in their make-up (Figure 2.6). This process is called *harmonization* of the scale. There are four kinds of triads, classified according to the nature of the intervals formed between the root and the third and between the third and the fifth. In the Table 2.3 we specify the intervals for each triad and we give an example of the scale and the degree that generate each type of triad.

Roman numbers identify not only the scale degree, but also the chord constructed upon that scale degree as a root. More details on the chord notation are in the Subsection 2.1.5

Table 2.3: *The four kinds of triads, classified according to the nature of the intervals formed between the root and the third and between the third and the fifth.*

Triad Name	Lower Interval	Upper Interval	Degree and Scale that produce the triad
Major Triad	maj 3rd	min 3rd	I of major scale
Minor Triad	min 3rd	maj 3rd	II of major scale
Diminished Triad	min 3rd	min 3rd	VII of major scale
Augmented Triad	maj 3rd	maj 3rd	III of minor (melodic) scale

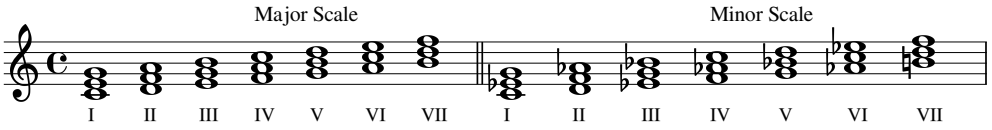

Figure 2.6: *Harmonization of major and minor (harmonic) scale.*

2.1.4 Chords with 4 or more notes

Once a triad is built, it is easy to extend it by superimposing another third. This is the most used way of creating chords with 4 notes. The chord so produced is called *seventh chord*, and the added note is called *seventh*. This kind of chord is widely used in jazz music, along with chords of 5, 6 and 7 notes. The number of possible combinations of thirds explodes while we add notes. However, since in this thesis we deal mainly with pop music, we consider a single seventh chord, called dominant seventh chord. It is built on the *V* degree of the major or minor scale and it is composed by the superposition of a major third and two minor thirds (Figure 2.7).

The operations on the triads (inversion of the chords and duplication of some notes) can be performed in the same way on the chords with 4 or more notes. In case of seventh chords, there are 3 possible inversions.

2.1.5 Harmonic Progression

Music can be considered as a temporal evolution of many musical parameters. It is common to refer to sequences of chords during a song as *harmonic progressions*. There are two common notations for chord progressions:

1. Sequences of pairs (root name, chord type). For example *D - E:min - G - A:min*.
2. Sequences of scale degrees, indicated by roman numbers (where the current scale must be defined explicitly). For example *I-II-V-VI* in the scale of D major.

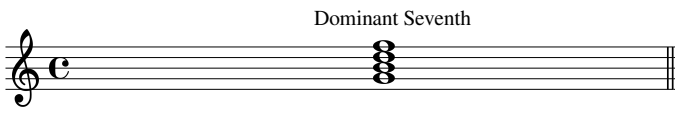

Figure 2.7: *A dominant seventh chord built on the V degree of the C major scale.*

The two notations have different advantages. The first notation allows to absolutely define a chord without needing further information about the musical context. It is the most simple notation and for this reason is widely adopted for jazz and modern music. The second notation, on the contrary, requires some experience in music theory, to understand exactly the kind of chord (minor, major, etc.) that is built on the specific root using the notes of the scale. This notation is useful for the analysis, since it is "pitch independent", i.e. it can be transposed to higher or lower pitches, by simply changing the scale of reference.

In tonal harmony different chords have different degrees of perceived stability. *I* is the most stable chord and *V* is the most unstable. Consequently each chord in a progression is invested with a particular role, called *harmonic function*. The basic chord progression is built starting from a stable chord, evolving towards more unstable chords and returning in the end to the stable chord.

There are many guidelines about how to manage harmonic progressions. For a complete discussion on this subject, we suggest [53] for classical/pop music and [38] for jazz music.

We saw in section 2.1.3 that there are many possible voicings for a chord. The way the different notes of chord voicings are connected in a progression is an import task in music composition, because it has a big impact on the way harmonic progressions are perceived. The study of this field is called *counterpoint*.

2.2 Mathematical Background

In this section we give some basic concepts of probability theory, information theory and machine learning that will be useful to understand the techniques that we will use for the generation of harmonic progressions.

2.2.1 Probability Theory

Probability theory is a mathematical framework for representing uncertain statements. It provides a mean for quantifying uncertainty, and axioms for deriving new uncertain statements. Nearly all activities require some ability to reason in the presence of uncertainty. In fact, beyond mathematical statements that are true by definition, it is difficult to think of any proposition that is absolutely true or any event that is absolutely guaranteed to occur.

There are three possible sources of uncertainty [20]:

1. Inherent stochasticity in the system being modeled. For example most interpretations of quantum mechanics describe the dynamics of subatomic particles as being probabilistic. It is also possible to create theoretical scenarios that we postulate to have random dynamics, such as a hypothetical card game where we assume that the cards are truly shuffled into a random order.
2. Incomplete observability. Even deterministic systems can appear stochastic when we cannot observe all of the variables that drive the behavior of the system. For

example, the behavior of a person who chooses to go outside or to stay at home depending on the weather, would seem stochastic, if it is not possible to observe the weather.

3. Incomplete modeling. When one uses a model that must discard some of the information observed, the discarded information results in uncertainty in the model prediction. For example, let us consider a robot that can exactly observe the location of every object around it. Since it has to discretize the space to save the information, the position of the objects for the robot become uncertain within the discrete cell.

A *random variable* is a variable that can take on different values randomly. It can be discrete or continuous. A discrete random variable is one that has a finite or countably infinite number of states. A continuous random variable is associated with a real value. In this thesis we will work with discrete random variables, since the events that we consider (chords, voicing, etc...) have a countable number of possibilities.

A *probability distribution* is a description of how likely a random variable or set of random variables is to take on each of its possible states. A probability distribution over discrete variables may be described using a *probability mass function (PMF)*. Specifically, a PMF maps from a state of a random variable to the probability of that random variable taking on that state. For example $p(z = x)$ defines the probability for the random value z to be equal to the value x. If the random value z we are referring to can be deduced from the context, we will write $p(x)$ to simplify the notation. Probability mass functions can act on many variables at the same time. Such a probability distributon over many value is known as a *joint probability distribution* $p(z = x, v = y)$. Again we will write $p(x, y)$ for brevity when the dependency on z and v is evident from the context. An example of PMF that we will use frequently is the *uniform distribution* over a discrete random variable z with k different states $x_1, x_2, ..., x_k$:

$$p(z = x_i) = p(x_i) = \frac{1}{k}, \quad \text{for } i \text{ from } 1 \text{ to } k \tag{2.2}$$

Sometimes it is known the probability distribution over a set of variables and one wants to know the probability distribution over just a subset of them (Figure 2.8). This value is known as the *marginal probability* distribution. For example, if there are discrete random variables z and v that can take values respectively from $X = x_1, x_2, ...$ and $Y = y_1, y_2, ...$ and $p(x, y)$ is known, it is possible to find $p(x)$ with the *sum rule*:

$$\forall x \in X, p(z = x) = \sum\nolimits_y \in Y p(z = x, v = y) \tag{2.3}$$

In many cases it is interesting to to know the *conditionl probability*, i.e. the probability of an event x, given that another event y has happened. It is denoted as $p(x|y)$. A useful formula that relates conditional and joint probability is:

$$p(y|x) = \frac{p(y, x)}{p(x)}, \quad \text{if } p(x) > 0 \tag{2.4}$$

An extension of the equation 2.4 is the so called *chain rule*. It considers a probability distribution over n random variables and explains how it can be decomposed into

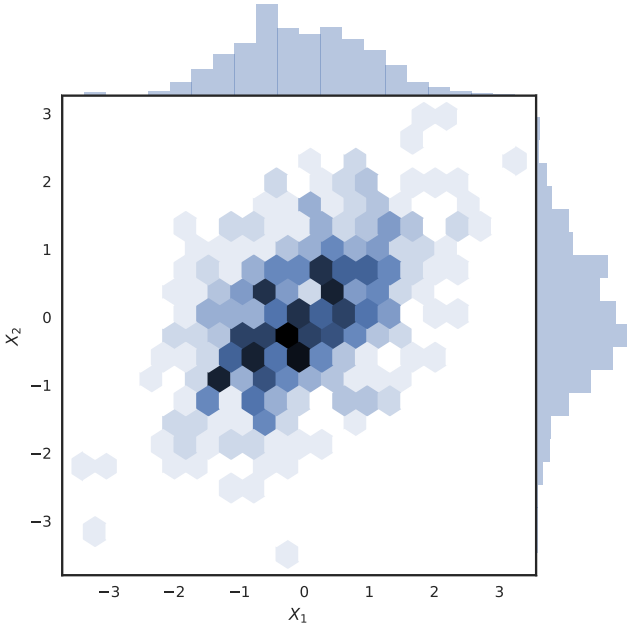

Figure 2.8: *The two marginal probabilities of a joint probability distribution of two variables $p(X_1, X_2)$, obtained with the sum rule.*

conditional distributions over only one variable:

$$p(x_1, ..., x_n) = p(x_1) \prod_{i=2}^{n} p(x_i|x_1, ..., x_{i-1}) \tag{2.5}$$

Two random variables x and y are *independent* if their probability distribution can be expressed as a product of two factors, one involving only x and one involving only y:

$$p(x, y) = p(x)p(y) \tag{2.6}$$

Two random variables x and y are *conditionally independent* given a random variable z if the conditional probability distribution over x and y can be factorize as follows:

$$p(x, y|z) = p(x|z)p(y|z) \tag{2.7}$$

The *expectation* or *expected value* of a function $f(x)$ with respect to a probability distribution $p(x)$ is the average value that f takes on when x is drawn from $p(x)$. For discrete variables this can be computed with a summation:

$$\mathbb{E}[f(x)] = \sum_{x} p(x)f(x) \tag{2.8}$$

The *variance* gives a measure of how much the values of a function of a random variable x vary, as different values of x from its probability distribution are sampled:

$$Var(f(x)) = \mathbb{E}[(f(x) - \mathbb{E}[f(x)])^2] \tag{2.9}$$

When the variance is low, the values of $f(x)$ cluster near their expected value.

The *covariance* gives some sense of how much two values are linearly related to each other, as different values of x from its probability distribution are sampled:

$$Cov(f(x), g(x)) = \mathbb{E}[(f(x) - \mathbb{E}[f(x)])(g(y) - \mathbb{E}[g(y)])] \tag{2.10}$$

High absolute values of the covariance mean that the values greatly change and are both far from their respective means simultaneously. If the sign of the covariance is positive, then both variables tend to take on relatively high values simultaneously. If the sign of the covariance is negative, then one variable tends to take on a relatively high value and the other takes on a relative low value.

Correlation is another measure that normalizes w.r.t. the scale of both variables, resulting in a bounded coefficient $[-1, 1]$. Two kinds of correlation used in statistics are *Pearson Correlation* and *Spearman Correlation*. Pearson correlation evaluates the linear relationship between two variables, while Spearman correlation assesses how well an arbitrary monotonic function can describe a relationship between two variables [26].

2.2.2 Information Theory

Information theory is a branch of applied mathematics that revolves around quantifying how much information is present in a signal . It was originally invented to study sending messages with a discrete alphabet over a noisy channel, such as communication via radio transmission. The basic intuition behind information theory is that learning that an unlikely event has occurred is more informative than learning that a likely event has occurred. For example a message saying "the sun rose this morning" is so uninformative as to be unnecessary to send, but a message saying "there was a solar eclipse this morning" is very informative [20]. Information theory quantifies information in a way that formalizes this intuition. Specifically:

- Likely events should have low information content (or no information content if the events are guaranteed to happen)
- Less likely events should have higher information content.
- Independent events should have addictive information. For example, if finding out that a tossed coin has come up as heads conveys a certain information x, finding out that it has come up as head twice should convey an information $2x$.

The *self-information* of an event x is defined with a formulation that satisfies all these properties:

$$I(x) = -\log p(x) \tag{2.11}$$

The choice of the base of the logarithm is not really important, however a common choice is to use base 2 logarithm. In this case information $I(x)$ is measured in bits.

Self-information deals only with a single outcome. It is possible to quantify the amount of uncertainty in an entire probability distribution using the *Shannon Entropy*:

$$H(x) = \mathbb{E}[I(x)] = -\mathbb{E}[\log p(x)] \tag{2.12}$$

Distributions that are nearly deterministic have a low entropy, while distributions that are closer to the uniform have high entropy.

If one has two separated probability distributions $p(x)$ and $q(x)$ over the same random variable x, it is possible to measure how different these two distributions are, using the *Kullback-Leibler (KL) divergence*:

$$D_{KL}(p \parallel q) = \mathbb{E}[log\frac{p(x)}{q(x)}] = \mathbb{E}[\log p(x) - \log q(x)] \tag{2.13}$$

The KL divergence is non-negative and it is 0 if and only if p and q are the same distribution.

A quantity that is closely related is the *cross-entropy*. For two distributions q and p, it measures the average number of bits needed to identify an event drawn from the set, if a coding scheme is used that is optimized for the distribution p, rather than the "true" distribution q. For discrete p and q it can be written as:

$$H(q, p) = -\sum_{x} q(x) \log p(x) \tag{2.14}$$

Unfortunately for many applications, like the estimation of the regularity of the models used in this thesis, the distribution q is unknown. In these cases, a typical approach is to use a Monte Carlo estimation of the true cross entropy [14]:

$$H_p(T) = -\frac{1}{|T|} \sum_{x \in T} \log_2 p(x) \tag{2.15}$$

2.2.3 Graphical models

Machine learning algorithms often involve probability distributions over a very large number of random variables. In order to reduce the complexity it is possible to split a probability distribution into a product of many factors. For example, let us suppose we have three random variables: a, b and c; let us suppose also that a influences the value of b and b influences the value of c, but a and c are independent given b. It is possible to represent the probability distribution over all three variables as a product of probability distributions over two variables:

$$p(a, b, c) = p(a)p(b|a)p(c|b) \tag{2.16}$$

This kind of factorization can greatly reduce the number of parameters needed to describe the distribution. Since each factor uses a number of parameters that is exponential in the number of variables in the factor, one can greatly reduce the cost of representing a distribution if one is able to find a factorization into distributions over fewer variables [14]. It is possible to describe this kinds of factorizations using directed acyclic graphs. The factorization of a probability distribution, represented with a graph, is called *Graphical Models* (Figure 2.9).

When one applies a graphical model to a problem in machine learning or pattern recognition, one will typically set some of the random variables to specific observed values,

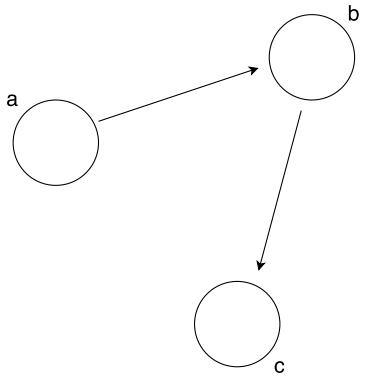

Figure 2.9: *A graphical model representing the probability distribution defined in the equation 2.16. The node represents the random variables and the arrows represent a variable conditioning another variable*

i.e. variables whose value is known. We denote such variables by shadowing the corresponding note.

Let us study the particular case of Figure 2.10, in order to derive some results that we will use later. By applying twice the chain rule 2.5 we obtain:

$$p(a, b, c) = p(a)p(c|a)p(b|c) \tag{2.17}$$

In the model (a), where none of the nodes is observed, if we test if a and b are independent, we obtain (marginalizing over c):

$$p(a, b) = p(a) \sum_c p(c|a)p(b|c) = p(a)p(b|a) = p(a)p(b|a) \tag{2.18}$$

which in general does not factorize into $p(a)p(b)$, so a and b are not independent. If instead we consider the model (b), where the node c is observed, we obtain:

$$\begin{aligned} p(a, b|c) &= \frac{p(a, b, c)}{p(c)} \\ &= \frac{p(a)p(c|a)p(b|c)}{p(c)} \\ &= p(a|c)p(b|c) \end{aligned} \tag{2.19}$$

from which we obtain the conditional independence property of a and b given c. The path from a to b is said to be *blocked* by the observed node c.

2.2.4 Machine Learning Basics

The field of machine learning is concerned with the question of how to construct computer programs that automatically learn from experience [45]. Mitchell suggests also a more precise definition for the concept of learning.

Definition 2.1. *A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at task in T, as measured by P, improves with experience E.*

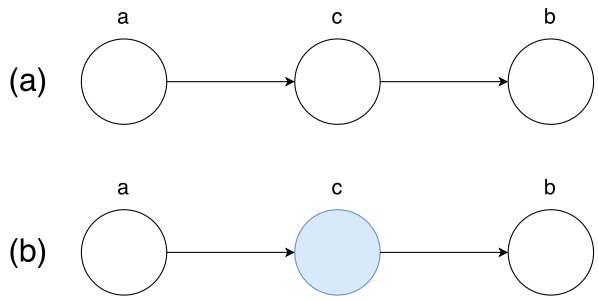

Figure 2.10: *Two different graphical models. The shadowed node in (b) represents an observed node.*

For example, it can be considered a machine learning system for handwriting recognition. In this case, the task T would be of recognizing and classifying handwritten words within images; the performance measure P would be the percentage of word correctly classified; the training experience E would come from a database of handwritten words with given labels.

The following sections provide an intuitive description and some examples of the different kinds of tasks, performance measures and experiences that can be used to construct a machine learning algorithm.

The Task, T. Machine learning allows us to tackle tasks that are difficult to solve with fixed programs written by human beings. Tasks are usually described in terms of how the machine learning system should process an input, i.e. a collection of features that have been quantitatively measured from some objects or events. We typically represent the input as a vector $x \in \mathbb{R}^n$, where each element x_i is a feature. The machine learning tasks can be clustered into three big groups:

- Classification: the computer is asked to specify which of k categories some inputs belongs to. This is usually accomplished by assigning to each input feature vector x a class from the k possible output classes (or a probability distribution over the output classes). A typical example is the handwriting recognition, where the input are the pixel of an image and the output is a letter from the alphabet.
- Regression: the computer is asked to predict a numerical value given some inputs. To solve this task, the learning algorithm is asked to output a function $f : \mathbb{R}^n \to \mathbb{R}$. An example of a regression task is the prediction of the value of a house, given a set of features such as the location, the number of rooms, the age and other factors.
- Clustering: the computer is asked to find a structure in a collection of unlabeled inputs. In other words, the task consists in organizing inputs into groups whose members are similar in some way. The exact meaning of "similar" or "dissimilar" inputs can change between different clustering algorithms. An example of clustering task can be to find similar groups of pixel in an image, in order to better perform image compression.

The Experience, E. The machine learning algorithms need a dataset of elements to be used as input. We call these elements data points. Usually, the bigger the dataset is,

the better the algorithm will accomplish its task; however there are algorithms that are made on purpose to work properly with bigger or smaller datasets. Machine learning algorithms can be broadly categorized a *supervised* and *unsupervised* by what kind of experience they are allowed to have during the learning process. "Supervised" means that both input and output are given, and the algorithm must learn the rules that relate an input with the corresponding output. In unsupervised learning only the inputs are given and the algorithm uses them to build an high-level representation of the data. For the purpose of this thesis, we will consider only supervised learning.

Performance Measure, P. In order to evaluate the abilities of a machine learning algorithm, quantitative measure of its performance must be designed. For the classification tasks, it is often measured the proportion of the input data for which the model produces the correct output. It is possible also to obtain similar results by measuring the proportion of input data that generates an incorrect output. The choice of performance measure is not straightforward. For example [14], when performing a transcription task, should we measure the accuracy of the system at transcribing entire sequences, or should we use a more fine-grained performance measure that gives partial credit for getting some elements of the sequence correct? When performing a regression task, should we penalize the system more if it frequently makes medium-sized mistakes or if it rarely makes very large mistakes? These kinds of design choices depend on the application.

Underfitting and Overfitting

In order for a machine learning algorithm to give meaningful results, it must perform well on previously unseen inputs, different from the inputs used to train the algorithm. Typically, a *training set* is available, and the *training error* can be reduced as an optimization process over the model parameters. What separates machine learning from simple optimization is that the error, generated by test data (data not used for training) must be low as well. The *test error* of a machine learning model is estimated by measuring its performance on a test set of inputs that were kept separate from the training set. The problem of improving the performances on the test set, if only the training set is observed, requires more assumptions to be solved: the train and test data must be considered as independent and identically distributed, generated by the same probability distribution (*i.i.d. assumption*).

With this notions it is possible to define the two central problems of machine learning: *underfitting* and *overfitting*. Underfitting occurs when the model is not able to obtain a sufficiently low error value on the training set. Overfitting occurs when the gap between the training error and the test error is too large. It is possible control whether a model is more likely to overfit or underfit by altering its *flexibility*, i.e. its ability to fit a variety of functions. Models with low flexibility may struggle to fit the training set and are not able to solve complex tasks. Models with high flexibility are prone to model noise and errors in the dataset that should not be part of the model. So we must carefully choose the right model flexibility. Unfortunately there is not a right choice for every dataset. The *No Free Lunch Theorem* [64] states that, averaged over all possible data

generating distributions, every classification algorithm has the same error rate when classifying previously unobserved data points. In other words, the most sophisticated algorithm that can be conceived has the same average performance (over all possible tasks) as a toy predictor that assigns all data points to the same class. This means that the goal of machine learning research is not to seek a universal learning algorithm or the absolute best learning algorithm. Instead, its goal is to understand what kind of distributions are relevant to the "real work" and what kinds of machine learning algorithms perform well on data drawn from the kinds of data generating distributions that are considered [20].

Regression Analysis

Regression analysis is one of the three tasks mentioned above. We provide here further details, since we will extensively use it in Chapter 4.

Suppose that for each value of a quantity x, another quantity y has a probability distribution $p(y|x)$. The mean value of this distribution, alternatively called the expectation of y, given x, and written $E(y|x)$, is a function of x and is called the regression of y on x. The regression provides information about how y depends on x [39]. The quantities x and y are respectively called *independent variable* and *dependent variable*. The simplest case is *linear regression*, where $E(y|x) = \omega_1 x + \omega_0$ for parameters ω_1 and ω_0. In case of *polynomial regressions*, the relationship between x and y is modeled as a nth degree polynomial in x. This can be seen in an informal way as a problem of curve fitting. In particular, one shall fit the data by using a polynomial function of the form:

$$y(x, \omega) = \omega_0 + \omega_1 x + \omega_2 x^2 + ... + \omega_M x^M = \sum_{j=0}^{M} \omega_j x^j \tag{2.20}$$

The values of the coefficients are determined by fitting the polynomial to the training set. This can be done by minimizing an *error function* that measures the misfit between the function $y(x, \omega)$, for any given values of $omega$ and the training set data points [7]. One simple choice of error function is given by the sum of squares of the errors between the predictions $y(x_n, \omega)$ for each data point x_n and the corresponding target values t_n, so that we minimize

$$E(\omega) = \sum_{n=1}^{N} (y(x_n, \omega) - t_n)^2 \tag{2.21}$$

This technique is called *least-square estimation*.

Robust Regression

Least-square estimation has bad performances when the error distribution is not normal, particularly when the errors are heavy-tailed [19]. In these situations there is the necessity to use another fitting criterion that is not as vulnerable as least-squares to outliers.

A possibility is to use the *Huber function* as error function. In contrast to the least-square function, Huber function assigns a weight to each observation, that declines

when $|e| > k$. The value k affects the performances of the regression: smaller values produce more resistance to outliers, but at the expenses of lower efficiency when the errors are normally distributed. Usually k is chosen as $k = 1.345\sigma$ (where σ is the standard deviation of the errors), since it produces 95-percent efficiency when the errors are normal, but still offers protection against outliers.

2.3 Related Work

The analysis of the relation between complexity and preference rating is not a particularly investigated field within the MIR community. Some works in this direction came from the field of psychology, and we present them in Section 2.3.1.

Then, we propose studies about the analysis of harmonic complexity and studies that consider perceptual descriptors such as "stability", "degree of completeness", "experienced tension", "music expectation", that can be considered related to our definition of harmonic complexity.

Lastly, we present also works that inspired the methodology for modeling of chord sequences, the analysis of the results and the design of the listening test.

2.3.1 Complexity and preference

In [2], Berlyne repetitively proposes to a subject black and white reproductions paintings with different levels of complexity: more complex paintings (crowded canvases with many human figures), less complex paintings (portraits of a single person), more complex nonrepresentational patterns (containing many elements), and less complex nonrepresentational patterns. He observes that, with repeated exposition, the less complex pictures were rated significantly more and more displeasing, while the ratings for the more complex pictures did not change significantly.

In [59], Skaife shows that preference tends, when "simple" popular music is presented, to decline with increasing familiarity. But with repetitive hearing of jazz, classical, or contemporary music that violates traditional conventions, there is normally a rise followed by a decline. Similar results were reached by Alpert [1] with rhythmic patterns.

In [3], Berlyne proves that the individual's preference for a certain piece of music is related to the amount of activity it produces in the listener's brain, which he refers to as *arousal potential*. In particular, there is an optimal arousal potential that causes the maximum liking, while a too low as well as a too high arousal potential results in a decrease of liking. He illustrates this behavior by an inverted U-shaped curve (Figure 1.2).

In [8], two groups of twenty college students, one with fewer than two years of musical training and the other with extensive musical backgrounds, rated their preferences for piano recordings of J.S. Bach's "Prelude and Fugue in C Major", Claude Debussy's "The Maid With the Flaxen Hair", Edvard Grieg's "Wedding Day at Troldhaugen", and

Pierre Boulez's "Piano Sonata No.1, 2nd Movement". Each piece had been ranked according to perceived complexity by seven music professors. When personal preference was plotted against complexity, an inverted U-shape curve was obtained.

2.3.2 Model-Based Symbolic analysis of Chord Progressions

One of the most intuitive approach is to analyze chord progressions, by trying to define explicitly the instinctive steps that a trained musician would perform. Therefore these techniques exploit knowledges and rules from music theory and propose algorithms to evaluate human perception of complexity and similarity.

In [41] [42], the authors propose a mathematical model based on Tonal Harmony (TH), in order to be able to use high level harmonic features in computer systems applications. In particular, they consider the harmony like a formal grammar, with some transformation rules between chords taken from music theory. If music obeys simple TH rules, they assign it a lower value, whereas if the rules are complex, or the harmony does not obey any known rules, they assign it higher values. The values of each chord transition are then combined and they define a new term, *harmonic complexity*. This feature is then tested on a music classification problem, proving that it is useful for Music Information Retrieval tasks.

Hanna et al. [25] in his *Chord Sequence Alignment System* (CSAS), compute the similarity value between two different chord sequences with an algorithm based on alignment. The similarity is computed with the recursive application of only three transformation rules: deletion of a symbol, insertion of a symbol, and substitution of a symbol by another. Each transformation has its own cost and, while the insertion and deletion costs are equal and constant, different functions for the substitution score (related to the roots, the basses, the types, the modes of the two chords) are proposed and compared.

De Haas [40] [16] presents a formalization of the rules of harmony as a Haskell (generalized) algebraic datatype. The Haskell advanced type system features make it possible to use datatype-generic programming to derive a parser automatically. Thus, given a sequence of chord labels, it is possible to derive a tree structure, which explains the harmonic function of the chords in the piece. The number of errors in creating the tree or the tree depth can be considered as a possible way to calculate the harmonic complexity by using tonal harmony, even though it was not the aim of the work. This system is then used on a corpus of 217 Beatles, Queen, and Zweieck songs, yielding to a statistically significant improvement for the task of machine chord transcription [15].

An interesting MIR system is the one developed by Pickens and Crawford [52] in order to retrieve polyphonic music documents by polyphonic music queries. The idea can be summarized in two steps. In the first part they preprocess the music score of each document in the collection in order to obtain a probability distributions over all chords, one for each music segment. In other words, instead of extracting chord labels from each score segment, they extract a 24 dimensional vector describing the similarity between the segment and every major and minor triad. In the second part they apply Markov modeling techniques, and store each document as a sequence of transitions (computed

with a fixed-order Markov Model) between these 24 dimensional vectors. Queries are then modeled using the same modeling technique of the documents and the Kullback-Leibler divergence is used as a ranking system to obtain the most similar document for the query.

2.3.3 Data-based Symbolic Analysis of Chord Progressions

Another possible approach is to put aside the prior knowledge from music theory and instead working with big dataset of annotated symbols (e.g. chords sequence or scores) in order to extract the rules that control music perception. This way of acting is supported by some psychology research [35] that pointed out that listeners appear to build on a set of basic perceptual principles that may adapt to different styles, depending on the kind of music they are exposed to.

In [63], the authors use multiple viewpoint systems and Prediction by Partial Matching to solve the problem of automatic four-part harmonization in accordance with the compositional practices of a particular musical era.

In [55], Rohrmeier et al. compare the predictive performance of n-gram, HMM, autoregressive HMMs, feature-based (or multiple-viewpoint) n-gram and Dynamic Bayesian Network Models of harmony. All this models use a basic set of duration and mode features. The evaluation is performed by using a hand-selected corpus of Jazz standards.

In [49], the authors propose to use a statistical-based data compression approach to infer recurring patterns from the corpus and show that the extracted patterns provide a satisfying notion of expectation and surprise. After this unsupervised step, they exploit the underlying algebra of Jazz harmony, represented as chord substitution rules, but they extract those rules from the dataset. The results prove that data-inferred rules correspond, in general, to the usual chord substitution rules of music theory textbooks.

In [58], the authors model chord sequences by probabilistic N-grams and use model smoothing and selection techniques, initially designed for spoken language modeling. They train this model with a dataset composed by 14194 chords from the 180 songs making up the 13 studio albums of The Beatles.

2.3.4 Mixed Symbolic Analysis of Chord Progressions

Paiement et al. [50] propose a work that use both the data-based and the model-based approach. They define a distributed representation for chords that gives a measure of the relative strength of each pitch class, taking into account not only the chord notes, but also the perceived strength of the harmonics related to every note (where the amplitude of the harmonics is approximated with a geometric decaying variable. This representation has the effect that the Euclidean distance between chords can be used as a rough measure of psychoacoustic similarity. In the second part of the paper the authors present a probabilistic "three shaped" graphical model of discrete variable, with three layers of hidden nodes and one of observed nodes (Figure 2.11). This structure can model chord progressions by considering different levels of context: variables in level 1 model the

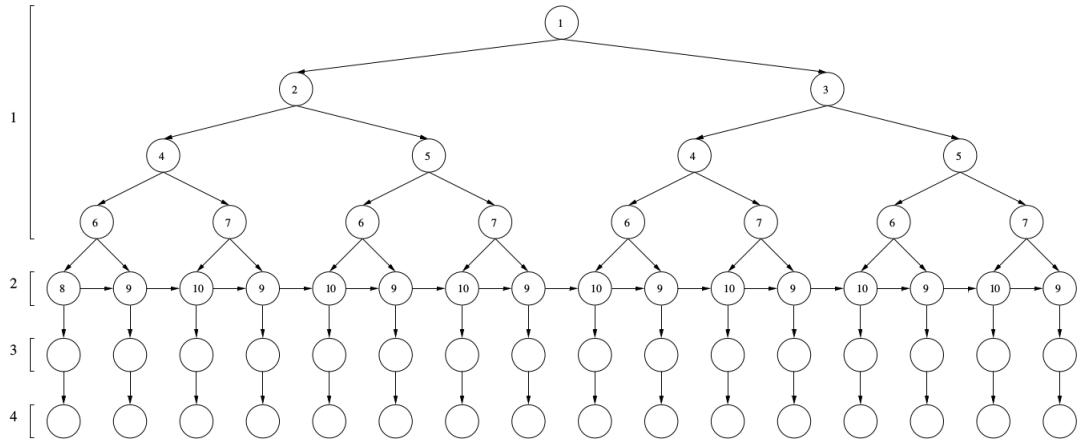

Figure 2.11: *A probabilistic graphical model for chord progressions. Numbers in level 1 and 2 nodes indicate a particular form of parameter sharing [Figure taken from [50]]*

contextual dependencies related to the meter, in level 2 local dependencies are taken into account and in level 3 it is used the distributed chord representation in order to substitute chords with "similar" chords. This last step gives a solution to the problem of unseen chords, since it just considers the "similar" chords to redistribute efficiently a certain amount of probability mass to those new events. This model is trained by using a dataset of 52 jazz standards; from the results it is possible to see that the three model performed better, compared with a standard HMM model trained on the same data.

2.3.5 Harmonic Complexity from Audio

Instead of starting from a symbolic representation, another approach is to use directly audio features.

In [44], the authors define a new meta-feature, *Structural Change*, that can be calculated from an arbitrary frame-wise basis feature, with each element in the structural change feature vector representing the change of the basis feature at a different time scale. Then they conducted a web-based experiment whose results suggest that the proposed feature correlates with the human perception of change in music. The relationship of structural change with complexity is hypothesized but left for future work.

In [62], the authors consider Beethoven's Sonatas and proceed to analyze harmonic complexity on different levels: chord level, fine structure, coarse structure, cross-work and cross-composer. Three different features are taken into account: Shannon entropy of the chroma vector, the quotient between the geometric and the arithmetic mean of the chroma vector, and harmonic similarity of the pitch classes computed on a reordered chroma vector to a perfect fifth ordering.

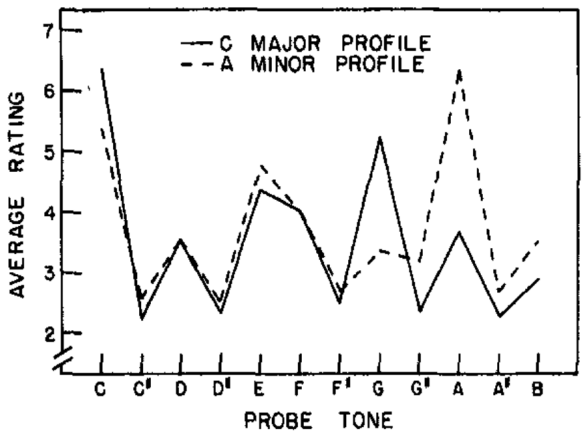

Figure 2.12: *The result of the probe tone experiment for C major and A minor keys. Listeners rated how much the specific tone sounds conclusive [Figure taken from [36]].*

2.3.6 Cognitive Approach to Music Analysis

Different contributions on how tonal organization affects the way music is remembered come from the field of psychology.

One of the first experiments is the *probe tone technique*, proposed by Krumhansl and Shepard [33], in order to quantify the hierarchy of tonal stability. In this study, an incomplete C major scale (without the final tonic, C) is sounded in either ascending or descending form in order to fix in the listener the perception of the C major key. The scale is then followed by one of the tones of the next octave (the probe tone). This experiment is repeated for every tone of the chromatic scale and the listener has to rate how well each tone completed the scale. The more musically trained listeners produced a pattern that is expected from music theory: the tonic was the most rated tone, followed by the fifth, third, the remaining scale tones, and finally the nondiatonic tones (Figure 2.12). In [36], Krumhansl and Kessler extend this method by using chord cadences and both major and minor scales as a context. A similar experiment [34] was conducted with the first movement of Mozart's piano Sonata K.282. The listeners had to adjust an indicator on the computer display to show the degree of experienced tension. The task was designed to probe three aspects of music perception: segmentation into hierarchically organized units, variations over time in the degree of experienced tension, and identification of distinct musical ideas as they are introduced in the piece. A deepening in this field can be found in [34].

In [5], the authors perform three listening tests to investigate the effect of global harmonic context on expectancy formation. Among the other results they demonstrate that global harmonic context governs harmonic expectancy and that neither an explicit knowledge of the Western harmonic hierarchy nor extensive practice of music are required to perceive small changes in chord function.

CHAPTER 3

Modeling Chord Progressions

THE easiest way to treat chord progressions would be simply to ignore the sequential aspects and to treat the observations as independent identically distributed (i.i.d.). The only information we could glean from data would be, then just the relative frequency of every single chord. However, if we consider tonal harmony rules, we can find out, for example, that after a dominant chord it is highly probable to find a tonic chord; or that after the sequence $I - II$ we can expect a dominant chord. Observing the previous chords is therefore of significant help in predicting the following chord.

We call this kind of data *sequential data* [7]. These are often produced through measurement of time series, for example the rainfall measurement on successive days at a particular location, the acoustic features at successive time frames used for speech recognition or, in this thesis, the chord sequence in a song.

It is useful to distinguish between stationary and non-stationary sequential distributions. In the stationary case, the data evolves in time, but the distribution from which it is generated remains the same. For the more complex non-stationary situation, the generative distribution itself is evolving with time. However, we will model the the rules for chord sequences generation as stationary, so here we consider only stationary sequential distributions.

To model the probability of a chord given the sequence of the previous chords $p(x_i|x_{i-1}, x_{i-2}..., x_{i-n})$, we choose to use three different models: Prediction by Partial Matching (section 3.1.2), Hidden Markov Model (section 3.1.3), and Recurrent Neural

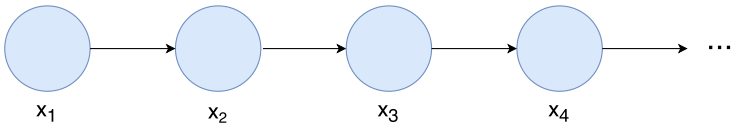

Figure 3.1: *A first order Markov Model. Each variable x_i is conditioned only on the value on the previous variable x_{i-1}.*

Network (section 3.1.5). Every model that we use can be considered as an extension of a single basic model for modeling sequential data: the so called Markov Model. So we dedicate the section 3.1.1 to explain it, in order to clarify the analysis of the more advanced models. Analogously we choose to illustrate the Feed Forward Neural Network (Section 3.1.4) for a better understanding of Recurrent Neural Networks.

The choice of using three different models has been done in order to use each model unique benefits. Specifically, it has been argued that the PPM model works really good with frequent sequences in the dataset, while the RNN model can "capture" rare behaviors and thus be more "creative" at sequence generation. The HMM stands on a middle-ground, since it models a hidden state, but in a non-distributed fashion. RNN and HMM model an hidden state that depends on all the precedent states (unbounded memory) while PPM has bounded memory [17].

3.1 Model Definition

3.1.1 Markov Model

It is possible use equation 2.5 to write a generic joint distribution for a sequence of T observations:

$$p(x_1, ..., x_T) = p(x_1) \prod_{i=2}^{T} p(x_i|x_1, ..., x_{i-1}) \tag{3.1}$$

Then, assuming that each of the observations is independent of all the previous observation, except the most recent, it is possible to simplify the equation 3.1 as following:

$$p(x_1, ..., x_T) = \prod_{i=2}^{T} p(x_i|x_{i-1}) \tag{3.2}$$

This is what is called a *first order Markov Model*, a type of probabilistic language model for predicting the next item a sequence. It is possible to draw it by using the notation introduced for graphical model in section 2.2.3 (Figure 3.1).

Since we are dealing with stationary sequential distribution, we constraints the transition probability distribution $p(x_i|x_{i-1})$ to be equal for each node of the network. For example, if that conditional distribution depends on adjustable parameters, then all of the conditional distributions in the chain will share the same values of those parameters.

Although this model has many possible uses, it is not generic enough to model sequences of events that depends on many event in the past. We introduce then the *n*-

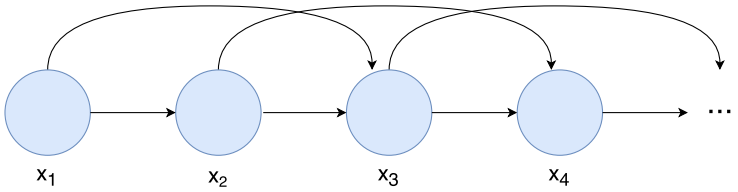

Figure 3.2: *A second order Markov Model. Each variable x_i is conditioned on the value of the two previous observations x_{i-1} and x_{i-2}.*

gram model, a model in the form of a (n-1)-order Markov model, i.e. the next element in a sequence depends on the previous $n-1$ elements.

$$p(x_i|x_{i-1}, x_{i-2}, ..., x_0) = p(x_i|x_{i-1}, ..., x_{i-n+1}) \tag{3.3}$$

By considering a song S consisting of a sequence of T chords labeled x_1,x_2,...,x_T, the *likelihood* of S is defined as:

$$p(S) = p(x_1, x_2, ..., x_{n-1}) \cdot \prod_{i=n}^{T} p(x_i|x_{i-1}, ..., x_{i-n+1}) \tag{3.4}$$

In order to compare the likelihood of chord sequences of different lengths and to simplify computations, the normalized negative log-likelihood is often used instead:

$$\bar{p}(S) = -\frac{logp(S)}{|S|} \tag{3.5}$$

and this quantity is expressed in bits per symbol. The perplexity of a set of songs H is then computed as:

$$\bar{p}(H) = -\frac{\sum_{S \in H} logp(S)}{\sum_{S \in H} |S|} \tag{3.6}$$

Maximum likelihood training aims to estimate the set of model probabilities such that $\bar{p}(H)$ is minimum over some training set H (distinct from the test set). This is achieved [58] by setting the transition probabilities to:

$$p(x_i|x_{i-1}, ..., x_{i-n+1}) = \frac{c(x_{i-n+1}, ..., x_i)}{\sum_{x_k \in X} c(x_{k-n+1}, ..., x_k)} \tag{3.7}$$

where $c(g)$ is the number of occurrences of the sequence g in the training set.

Such models have been used for many types of data, such as melody [13] and spoken language [10]. The first attempts to model chord sequences via this approach [61] [43] achieved limited success, due to fixed model inputs and parameters and to overfitting issues. Better results have been obtained using smoothing techniques [58] originally designed for spoken language modeling.

3.1.2 Prediction by Partial Matching - PPM

Prediction by Partial Matching is an extension of the N-gram model that defines a more sensible behavior in case of unobserved sequences (zero frequency problem).

When the training set is small compared to the number of possible N-grams, it is likely to find, in the test set, sequences that were never observed. By using the equation 3.7, we should assign to those sequences a probability 0, but this will then conflict with observation of the sequence in the test set. In order to solve this problem, it is possible to start from the idea that, even if we have never observed the sequence S, we may have observed the last part part of S, and this could suggest something about the sequence probability. This may appear clearer by reasoning with a language example: we may have never seen the sequences "John likes to drink water" and "John likes to eat water", but we may have observed the sequence "like to drink water"; so given the observed shorter sentence, we would like to assign an higher probability to "John likes to drink water" with respect to "John likes to eat water". This is called "Prediction by Partial matching" and, starting from this idea, it is possible to rewrite the equation 3.7 by taking into account also lower order n-grams [32] in order to perform a sort of *smoothing* of the probability distribution:

$$p(x_i|x_{i-1},...,x_{i-n+1}) = \\ = \begin{cases} \alpha(x_i, x_{i-1},...,x_{i-n+1}), & \text{if } c(x_{i-n+1},...,x_i) > 0 \\ \gamma(x_i,...,x_{i-n+1})p(x_i|x_{i-1},...,x_{i-n+2}), & \text{otherwise} \end{cases} \tag{3.8}$$

where $\alpha(x_i|x_{i-1},...,x_{i-n+1})$ is an estimate of the probability of an already seen n-gram and the escape probability $\gamma(x_{i-n+1},...,x_i)$ represents the probability mass assigned to all symbols that have not been observed in the training set.

PPM optimizations

There are many optimization designed to improve the performances of PPM models. We present here two common techniques to improve the probabilities estimated by PPM:

- *Exclusion*: a technique that excludes predictions found at a higher order when calculating the probabilities for lower-order contexts [12]. Let us explain it with a short example: suppose we trained our model on the progression (C, $Dmin$, G, $Dmin$, F, C, $Dmin$) with a 2-gram model. If we then encounter the chord F after this progression, it is not predicted by the current $k = 2$ context, because we have never encountered the triple (C, $Dmin$, F). So we go back to the $k = 1$ context and we find two occurrence of $Dmin$ followed by another chord. However, we can exclude the sequence ($Dmin$, G) from our computations, because the chord G cannot possibly occur, since if it did, it would have been encoded at the $k = 2$ level.
- *Interpolated Smoothing*: a technique that proposes an improvement of the quation 3.8:

$$p(x_i|x_{i-1},...,x_{i-n+1}) = \\ = \max\Big(\alpha(x_i, x_{i-1},...,x_{i-n+1}), 0\Big) + \gamma(x_i,...,x_{i-n+1})p(x_i|x_{i-1},...,x_{i-n+2}) \tag{3.9}$$

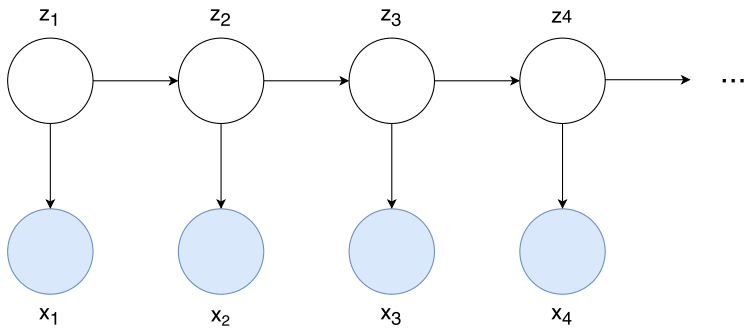

Figure 3.3: *A graphical representation of a Hidden Markov Model: a Markov chain of latent variables where the probability of each observation is conditioned on the state of the corresponding latent variable.*

With this technique [51], the probability of an n-gram is always estimated by recursively computing a weighted combination of the $(n-1)$th order distribution with the $(n-2)$th.

3.1.3 Hidden Markov Model - HMM

Suppose that we want to model a sequence of events, whose next event depends on many events in the past. We saw in section 3.1.1 that we can use a n-gram for this kind of problems, and increase the value of n. However we must pay a price for the increased flexibility, because the number of parameters grows exponentially with n. In particular, with K possible states, a model that looks for the past n events will have $K^{n-1}(K-1)$ parameters [7]. So, for larger values of n, this approach can be impractical.

If we want to build a model for sequences that is not limited by the Markov assumption to any order, and yet can be specified using a limited number of free parameters, a better approach is to use a Markov chain of latent variables, with each observation conditioned on the state of the corresponding latent variable (Figure 3.3). If the latent variables are discrete, we call this *Hidden Markov Model* (HMM).

An HMM is characterized by the following: [54]:

1. N, the number of states in the model. Although the states are hidden, for many applications there is often some physical significance attached to the states or to sets of states of the model. For example in our work, hidden states could be related to some specific tonal context [17] (although for a high number of hidden state, this intuition can be not significant), like dominant chords, sub-dominant chords, etc... We denote the hidden variable alphabet as $q = \{q_1, q_2, ..., q_N\}$ and the state at time t as z_t.

2. M, the number of distinct observation symbols per state, i.e. the discrete alphabet size. The observation symbols correspond to the physical output of the system being modeled. In our work, the observation symbols are all the chords that are present in the database of songs. We denote the individual symbols as $v = \{v_1, v_2, ..., v_M\}$ and the symbol at time t as x_t

3. $A = a_{ij}$, the *state transition probability distribution*, where:

$$a_{ij} = p(z_{t+1} = q_j | z_t = q_i), \quad 1 \leq i, j \leq N \tag{3.10}$$

Since a_{ij} are probabilities, they satisfy the equation: $0 \leq a_{ij} \leq 1$ with $\sum_i a_{ij} = 1$, so the matrix A has $N(N-1)$ independent parameters.

4. $B = b_j(k)$, the *emission symbol probability distribution*, where:

$$b_j(k) = p(x_t = v_k | z_t = q_j), \quad 1 \leq j \leq N \wedge 1 \leq k \leq M \tag{3.11}$$

The same properties written for transition probabilities are valid also for emission probabilities.

5. $\pi = \pi_i$, the initial state distribution, where:

$$\pi = p(z_1 = q_i), \quad 1 \leq i \leq N \tag{3.12}$$

where $\sum_i \pi_i = 1$.

Again we focus our attention on *homogeneous* models, for which all the conditional distributions governing the latent variables are encoded in the same matrix A and all the emission distributions are described by the same matrix B. The joint distribution over both latent and observed variables is then given by [7]:

$$p(X, Z; \theta) = p(z_1|\pi) \left[\prod_{t=2}^{T} p(z_t | z_{t-1}; A) \right] \left[\prod_{\tau=1}^{T} p(x_\tau | z_\tau; B) \right] \tag{3.13}$$

where $X = \{x_1, ..., x_T\}$, $Z = \{z_1, ..., z_T\}$, and $\theta = \{\phi, A, B\}$ denotes the set of parameters governing the model.

We can gain a better understanding of the Hidden Markov Model by considering it from a generative point of view [7]. We first choose the initial latent variable z_1 with probabilities governed by the parameter π and then sample the corresponding observation x_1 from v, using the observation probability $b_j(1)$. Then we choose the state of the variable z_2 according to the transition probabilities $a_{ij} = p(z_2|z_1)$ since we already know the value of z_1. From the value of the hidden state z_2 we sample the symbol x_2. From z_2 we can generate z_3 and so on.

A useful property of HMM is that, when we use it to model an already existing sequence (during the training phase) the hidden node probability z_t will depend on all the previous hidden state $z_{t-1}, z_{t-2}, ...$ since there are no observed nodes that block the path. In our work this means that the model will "have memory" of all the past events. Instead, when we use the model as a generator, as explained above, we will know the past hidden nodes, so we have a blocked path (Equation 2.19). This allow us to generate the next hidden state z_{t+1} relatively easily, taking in account just the current state z_t, since $z_{t+1} \perp\!\!\!\perp z_{t-1} \mid z_t$.

It is important to emphasize that the application of HMM in this thesis, for modeling chord progression, is different from the use of HMM for chord recognition (e.g. from audio). In chord recognition, the hidden nodes are the actual chords and the observed nodes are the audio feature. Here the observed nodes are the actual chords from the input score and the hidden nodes can be related to some abstract harmonic context, as stated before.

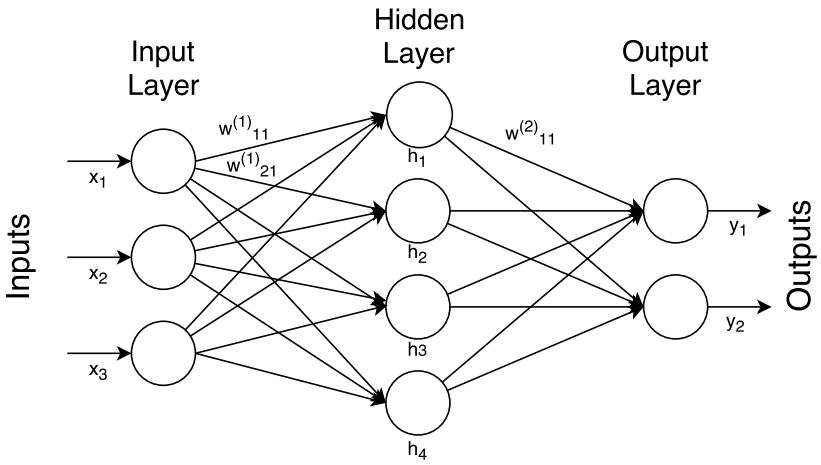

Figure 3.4: *A simple feedforward neural network with a single hidden layer. The input layer has 3 nodes, the hidden layer has 4 nodes and the output layer has 2 nodes. This network can be used to perform a classification task with $k = 2$ classes.*

3.1.4 Feedforward Neural Network

Feedforward neural networks are of extreme importance to machine learning practitioners. They form the basis of many important commercial applications. For example, the convolutional networks used for object recognition from photos are a specialized kind of feedforward network. The term *neural network* has its origins in attempts to find mathematical representations of information processing in biological systems. Even if the claim of their biological plausibility may have been exaggerated, the term *network* is still meaningful, since we compose together many different functions with a structure that can be represented with a direct acyclic graph.

To understand the idea at the base of feedforward neural networks, we can start by thinking about the limitations of linear models, such as logistic regression and linear regression. How can we overcame the defect that the model capacity is only limited to linear functions? The idea is to apply the linear model not to the input x itself, but to a transformed input $\phi(x)$, where ϕ is a non-linear transformation. The question is then how to choose the mapping ϕ. Until the advent of what is called *deep learning*, the typical approach was to manually engineer ϕ. Unfortunately this approach requires a lot of human effort for each single task. The strategy of deep learning is, instead to decide just the general function family of ϕ and to learn its exact parameters using machine learning techniques.

For example, consider the task of approximate a function f^*, for instance the classification function $y = f^*(x)$ that maps an input x to a distribution y. A FNN defines a mapping:

$$y = f(x; \theta, \omega) = \phi(x; \theta)^T \omega \tag{3.14}$$

where there are parameters θ that are used during the training phase to learn ϕ from a broad class of functions, and parameters ω that map from $\phi(x)$ to the desired output [20].

For a more practical understanding, we now describe the functioning of the simple

FNN of Figure 3.4. First we construct M linear combinations of the input variables $x = x_1, ..., x_D$ ($D = 3$ in the example)in the form:

$$a_j = \sum_{i=1}^{D} \omega_{ji}^{(1)} x_i + b_j^{(1)} \tag{3.15}$$

where $j = 1, ..., M$ ($M = 4$ in the example). We shall refer to the parameter $\omega_{ji}^{(1)}$ as *weights* and the parameter $b_j^{(1)}$ as *biases*. The quantities a_j are known as *activations*. Each of them is then transformed by using a differentiable, nonlinear *activation function* $f(\cdot)$ to give:

$$h_j = f(a_j) \tag{3.16}$$

These quantities are again linearly combined with the equation:

$$a_k = \sum_{l=1}^{M} \omega_{kl}^{(2)} z_j + b_k^{(2)} \tag{3.17}$$

and, finally, the output unit activations are transformed using an appropriate activation function to give a set of network outputs y_k.

This chain structure is the most commonly used in neural networks. It is possible to generalize this model adding multiple layers in the chain. The overall length of the chain gives the *depth* of the model. A common denomination for the generic model is to call *input layer* and *output layer* respectively the first and the last layers, and *hidden layers* all the layers in between. This name is due to the fact that the training data does not show the desired output for each of these layers, but instead it is the training algorithm that must decide how to use them to produce the correct final output. Another generalization is to change the number of nodes for each layer (the *width* of the model). Rather than thinking of the layer as representing a single vector-to-vector function, it is also possible to think of the layer as consisting of many *units* that act in parallel, each representing a vector-to-scalar function: each unit receives inputs from many other units and computes its own activation value. The propagation of inputs up to the hidden units and then to the output layer is called *forward propagation*.

The choice of the family of the activation function is determined by the nature of the data and the assumed distribution of target variables. For example, for standard regression, the activation function is the identity ($y_k = a_k$), for binary classification it is used the sigmoid function, while for multiclass classification, with $K > 2$ output classes, is used a softmax (nonlinear) activation function in the form:

$$\begin{aligned} softmax: \quad & p(C_k|x) = \frac{\exp(a_k)}{\sum_j \exp(a_j)} \\ \log softmax: \quad & \log p(C_k|x) = a_k - \log \sum_j \exp(a_j) \end{aligned} \tag{3.18}$$

The training algorithm when working with NN is the so-called *backpropagation*. During training, the forward propagation produces a scalar prediction, which is compared

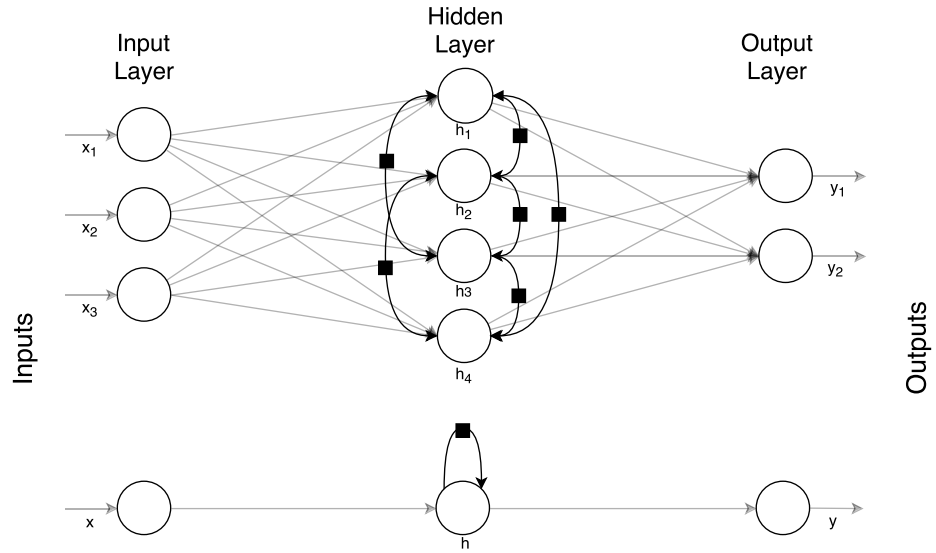

Figure 3.5: *A graphical representation of a simple RNN with a single hidden layer. The black square indicates a delay of a single timestep. We can see that both the input and the previous hidden state participate at the computation of the current hidden layer. In the bottom part, a compact representation, where all the nodes are represented in a single node for each layer.*

to the target value in orded to obtain a cost $J(\theta)$. The *back-propagation* algorithm allows the information from the cost to flow backwards through the network, in order to compute the gradient to use for the optimization of the parameters. Computing an analytical expression for the gradient is straightforward, but numerically evaluating such an expression can be computationally expensive. The strength of the back-propagation algorithm is that it can do this task by using a simple and inexpensive procedure [20]. The work presented in [57] was one of the first successful experiments with this algorithm and contributed in initiating a very active period of research in multi-layer neural network. The other big improvement, that made neural network a competitive model, was presented in [28] [27]. Hinton et al. showed how a many-layered feedforward neural network could be effectively pre-trained, one layer at time as an unsupervised restricted Boltzmann machine, and then fine-tuned by using supervised back-propagation.

3.1.5 Recurrent Neural Network - RNN

Recurrent Neural Networks extend standard feed-forward deep neural networks, by making them scale to much longer sequences than would be practical for "standard" neural networks.

The basic idea is to share parameters across different part of the model, thus allowing a "memory" of previous inputs to persist in the network's internal state, and thereby influence the network output. This makes it possible to extend and apply the model to sequences of different length and to generalize across them. In order to achieve that, cyclical connections are allowed (Figure 3.5), thus making the model auto-regressive.

A useful way to visualize RNNs is to consider the update graph formed by "unfolding" the network along the input sequence (Figure 3.6). Viewing RNNs as unfolded

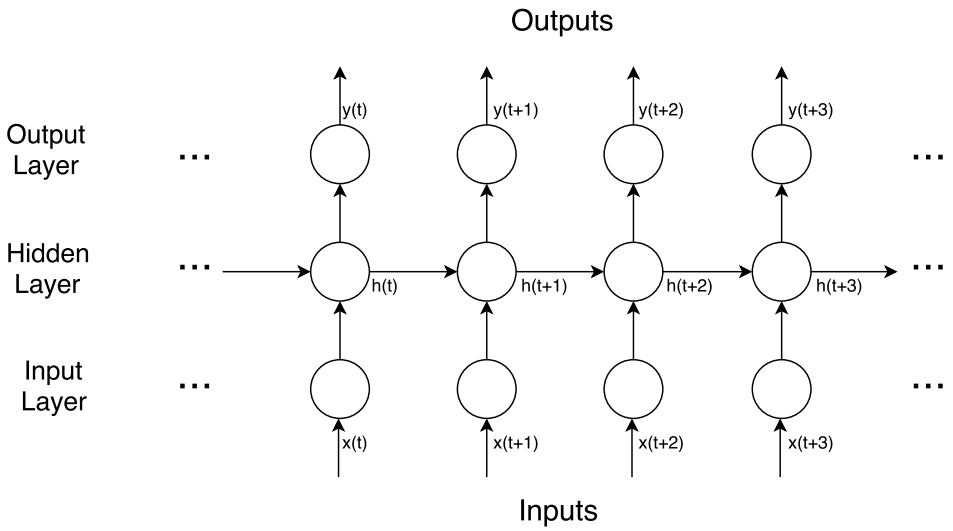

Figure 3.6: *The unfolded graphical representation of a simple RNN with a single hidden layer where it is clear that the current hidden state $h(t)$ is participating in the computation of the next hidden state $h(t+1)$. Note that the unfolded graph contains no cycles.*

graphs makes it easier to generalize to networks with more complex update dependencies.

Using RNN it is possible to overcame the limitation of FNN of accepting only fixed-length sequences as input. Since RNN models the concept of time, sequences of arbitrary length can be used as a context, similarly to HMM. However, differently from HMM, RNNs have a distributed state and therefore have significantly larger and richer memory [23].

Allowing cyclical connections can be translated in mathematical formulas by modifying the equation 3.15 as follows [24]:

$$a_j(t) = \sum_{i=1}^{D} x_i(t) v_{ji} + \sum_{l=1}^{M} h_l(t-i) u_{jl} + b_j \tag{3.19}$$

where v_{ji} are the weights from the input layer to the hidden layer and u_{jl} are the weights from the previous hidden layer to the current hiddent layer. The equation 3.17 is still valid for RNN.

Deep Recurrent Network

The computation in the RNNs that we analyzed can be decomposed into three stages [20]:

1. from the input to the hidden state;
2. from the previous hidden state to the next hidden state;
3. from the hidden state to the output.

Figure 3.7: *A graphical representation of a deep RNN where the hidden recurrent state is broken down into many layers organized hierarchically.*

Would it be advantageous to increase the number of hidden layers? Experimental evidence ([22]) suggest so, in agreement with the idea that we often need enough depth in order to perform the mapping that we need for our applications. From the many possibilities to increase the depth of a RNN, we choose the simple method exposed in Figure 3.7. We can think of the hidden layers lower in the hierarchy as playing a role of transforming the input into a representation that is more suited to be processed by hidden layers at higher levels.

Long Short-Term Memory

As discussed in the previous section, an important benefit of recurrent neural networks is their ability to use contextual information when mapping between input and output sequences. Unfortunately, for standard RNN architectures, the range of contexts that can be accessed in practice is quite limited. The problem is that the influence of a given input on the hidden layer, and therefore on the network output, either decays or blows up exponentially as it cycles around the network recurrent connections. This effect is often referred to in the literature as the vanishing gradient problem [21]. Numerous attempts were made in the 1990s to address the problem of vanishing gradients for RNNs. We choose to use one of the most common architectures, named *Long-Short-Term Memory (LSTM)* architecture [29].

The LSTM architecture consists of a set of recurrently connected subnets, known as memory blocks. These blocks can be thought of as a differentiable version of the memory chips in a digital computer. Each block contains one or more self-connected memory cells and three multiplicative units: the input, the output and forget gates. They provide continuous analogues of write, read and reset operations for the cells (Figure 3.8). An LSTM network works in the same way as a standard RNN, except that the summation units in the hidden layer are replaced by memory blocks. The multiplicative gates allow LSTM memory cells to store and access information over long periods of time. For example, as long as the input gate remains closed (i.e. has an activation near 0), the activation of the cell will not be overwritten by the new inputs arriving in the network, and can therefore be made available to the net much later in the sequence, by opening the output gate.

3.2 Dataset

The dataset was obtained and processed in [17]. We report here the salient details, for reading convenience.

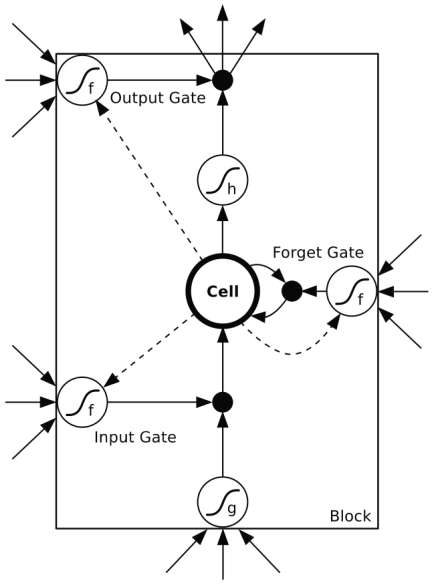

Figure 3.8: *The three gates are nonlinear summation units that collect activations from inside and outside the block, and control the activation of the cell via multiplications (small black circles). The input and output gates multiply the input and output of the cell while the forget gate multiplies the cell previous state. Note that the block has four inputs but only one output [Figure taken from [23]].*

The dataset was obtained from *ultimate-guitar.com*, a website that presents itself as "Your #1 source for chords, guitar tabs, bass tabs, ukulele chords, guitar pro and power tabs". Basically the website allow everyone to upload chord annotations, and contains a huge collection of user annotated songs. Overall the dataset contains 26,545,277 chord occurrences stemming from 441,2924 songs by 41,562 authors. Style tags show a significant prevalence of Rock, Pop and Alternative. Since the songs are not annotated by fields-experts, the dataset is quite noisy, but the authors argued that the benefits of such a large data set outweighs any effect of the noise in the data [17].

In order to make the parsed chords sequences usable for the models, it was necessary to make them pass through a cleaning process:

1. Map the equivalent chord type labels to the same label. This step was necessary since the original annotations did not provide a consistent syntax, e.g. the chord composed by the pitches C-E-G-B can be annotated as $Cmaj7$, $C7+$, $C\ \triangle\ 7$, $C\ \triangle$.

2. Map harmonic equivalent chords to the same label. E.g. from the rules of Tonal Harmony, we know that considering its harmonic function, the chord $D13$ can, in a first approximation, be consider equivalent to $D9$, and $D7$.

3. Ignore the bass annotations. Approximately 2% of the chords were annotated with a bass note different from the root (e.g. C/E), so the authors decided to remove it since the complexity introduced by a much larger alphabet was not enough compared with the information contained in this annotation.

4. Map some rare chord types to "similar" frequent types. This was the most subjective decision of the entire process, but, again, it was necessary to reduce the

Table 3.1: *Mapping from "rare" chord types to more frequent types in order to reduce the number of symbols in the model*

source	target
Z:aug	Z:maj
Z:dim	Z:min
Z:sus4	Z:(5)
Z:sus2	Z:(5)
Z:7sus4	Z:7
Z:hdim7	Z:min

complexity of the model. In the table 3.1 we show the mapping of these labels.

5. Decouple the harmonic function from the specific pitch. Keeping the exact chord fundamental pitch is redundant; what it is really meaningful is the interval from the chord fundamental to the local tonic. In order to perform this step it was necessary to perform the tonic identification of each song and then translate every song chord so that every song tonic corresponds to the same pitch. To do that it was used the tonic identification system developer by Di Giorgi [17].

After this process, our dataset is composed by 4 chord types (maj,min,7,5) and 12 tonic-fundamental intervals (overall 48 symbols). The first three types were chosen because they are, as expected, the most frequently occurring in the dataset (89% of all occurrences after step 3), while (5) was retained because the mapping to one of the other three types is not possible without prior information about the tonality.

3.3 Model Implementation and Training

In this section we explain how each model is implemented in order to better accomplish the task of modeling chord progressions. Then we discuss some theoretical problems regarding the training phase and we briefly present the training phase performed by Di Giorgi [17]. A detailed explanation about the training phase can be found in his paper.

The training of the models is basically an optimization problem over a specific evaluation function. A simple and efficient evaluation function is the cross-entropy (Eq. 2.14). In this thesis, q is the distribution of chords in any corpus, and p is the distribution of chords as predicted by the models. So the lower the cross-entropy, the better we can expect the language model to predict the next chord. Unfortunately, the distribution q is unknown, so we will use its Monte Carlo estimation (Eq. 2.15) where the training (and test set) set is treated as samples from $q(x)$. The sequence used for training are sequences of chords symbols (or, using the musical notation of section 2.1.5, harmonic progressions) whose specific format is described in section 3.2.

In order to give an insight on how the models will generalize to an independent dataset and limit overfitting problems, it was evaluated the variability of the cross-entropy estimates using a *k-fold cross-validation* [20], with $k = 5$. This method consists in partition the original data set into k non overlapping, equal sized subset. A single subset is retained as the validation data for testing the model and the remaining $k - 1$ are used

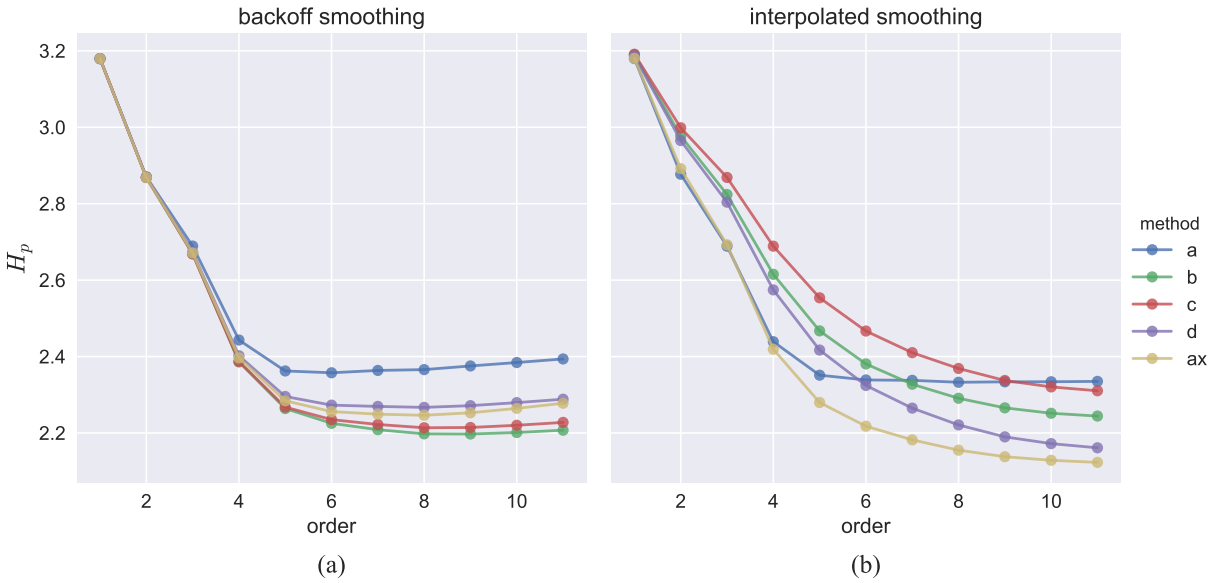

Figure 3.9: *A comparison between the five smoothing methods of Table 3.2 with Backoff Smoothing (a) and Interpolated Smoothing (b). Both figures use Exclusion. Error bars computed by 5-fold cross validation are not visible, being two orders of magnitude smaller than the data.*

as training data. This step is then repeated k times, with each of the k subset used only once for validation. The k results are then averaged to produce a single estimation.

3.3.1 Prediction by Partial Matching - PPM

With reference to Eq. 3.8, for each context, one must allocate a probability $\gamma(x_i, ..., x_{i-n+1})$ to the event that a novel sequence occurs, but it is difficult to image a rationale for optimal choice of this probability [11]. Some solutions in the context of Markov models have been proposed by [11], [46], [47]. As noted by all these authors, in absence of a priori knowledge, there seems to be no theoretical basis for choosing one solution over another. So it is necessary to take a pragmatic approach and test the many solutions in the particular context of use.

The different methods to obtain $\hat{c}(x)$ and ξ are listed in the Table 3.2. For more information about this process, we refer to [17].

Table 3.2: *List of escape methods tried during the training phase of PPM model [Table taken from [17]].*

method	escape if	ξ	$\hat{c}(x)$	$sum_{x \in X} \hat{c}(x)$
A [11]	$c(x) = 0$	1	$c(x)$	N
B [11]	$c(x) \leq 1$	N_{2+}	$c(x) - 1$	$N - N_1 - N_{2+}$
C [46]	$c(x) = 0$	N_{1+}	$c(x)$	N
D [30]	$c(x) = 0$	$0.5N_{1+}$	$c(x) - 0.5$	$N - 0.5N_{1+}$
AX [47]	$c(x) = 0$	$N_1 + 1$	$c(x)$	N

The cross-entropy results achieved by the methods in Table 3.2 are shown in Figure 3.9. All the methods used the Exclusion technique. The variations on H_p caused by 5-fold

cross validation are two orders of magnitude smaller than the data, meaning that the risk of overfitting is very low.

3.3.2 Hidden Markov Model

The problem of training an Hidden Markov Model, can be explicitly defined as the problem of adjusting the model parameters $\theta = (A, B, \pi)$, given the observation sequence $x = \{x_1, x2, ..., x_T\}$. The goal is to maximize the likelihood function (obtained by marginalizing the equation 3.13 over the latent variable):

$$p(X; \theta) = \sum_{Z} p(X, Z; \theta) \tag{3.20}$$

Unfortunately it is possible to simply treat each of the summations over z_t independently, because the joint distribution does not factorize over t. Nor can we perform the summation explicitly, because there are T variables to be summed over, each of which has N states (notation of section 3.1.3), resulting in a total of N^T terms [7].

An efficient technique to solve this optimization problem is the *Expectation-Maximization* (EM) algorithm. It start with some initial selection for the model parameters, which are denoted by θ^{old}. The algorithm proceeds in two steps applied iteratively to each observation in the dataset until convergence of the parameters:

1. E step: it takes the values of the parameters and evaluates the posterior distribution of the latent variables $p(Z|X, \theta^{old})$. It then use it to evaluate the expectation of the logarithm of the complete-data likelihood, as a function of the parameters θ, to give the function:

$$Q(\theta, \theta^{old}) = \sum_{Z} p(Z|X, \theta^{old}) \log p(X, Z, \theta) \tag{3.21}$$

2. M step: it finds the new value of θ that maximizes $Q(\theta, \theta^{old})$.

The EM algorithm must be initialized by choosing starting values for π and A. But elements of π and A that are set to zero initially, will remain zero in subsequent EM updates [7]. A typical initialization procedure is to select random numbers, making sure that they respect the summation and the non negative constraints associated with their probabilistic interpretation.

The number of possible hidden nodes N (i.e. the size of the alphabet), is a parameter that sensibly affects the performance of the model. A low number will lead to poor data fitting and to a reduction of the importance of the context (in the extreme case of $n = 1$ the model will become an 1-gram model, with no memory). On the other side, choosing too high a number may cause problems of overfitting and a degradation of the performances with the real world-applications.

The size of the alphabet Q was varied logarithmically from 1 to 1000, and to use 100 iterations of EM algorithm. The number of parameters Λ used by the model, can be calculated as:

$$\Lambda = (N - 1) + N(M - 1) + N(N - 1) \tag{3.22}$$

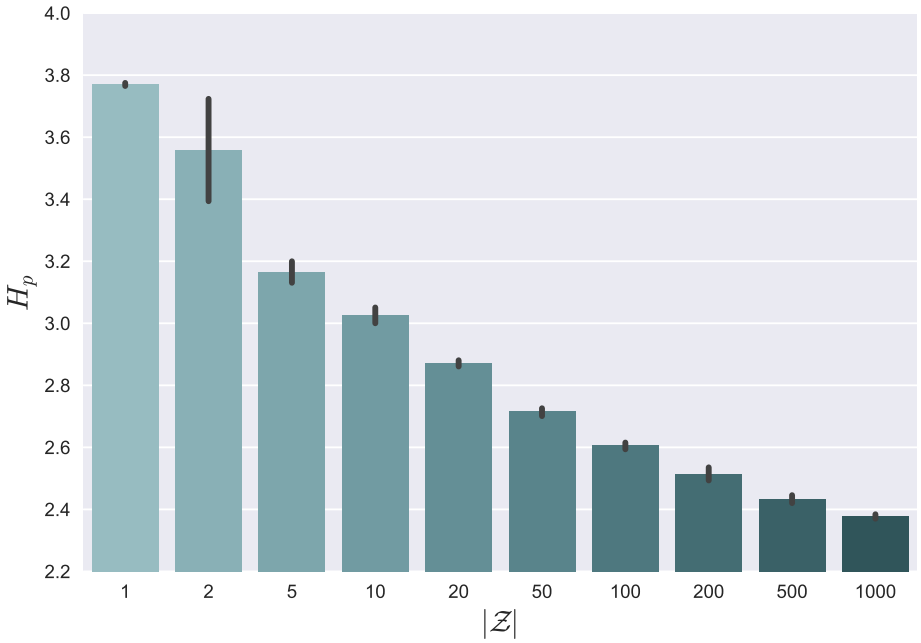

Figure 3.10: *The cross-entropy of the HMM model with different hidden node alphabet sizes N. Error bars are computed using 5-fold cross validation [Figure taken from [17]].*

The Figure 3.11 shows the cross-entropy of the trained models. As we can expect, it decreases while increasing N. No overfitting problems are found with the used alphabet sizes ($N \leq 1000$).

3.3.3 Recurrent Neural Network

In order to model harmonic progressions with RNNs, each chord was transformed in a binary vector of 48 entries (48 is the dimension of the alphabet of chords) where just the corresponding entry is equal to 1. The LSTM technique is used to enable the model to consider a longer context and the gradients are computed with backpropagation. Since this is a problem of multi-class classification with $k > 2$, the output layer function f is a softmax.

In his paper [17], Di Giorgi experimented with different configurations, by varying the number of hidden layers and the number of units. The results are in the Figure 3.11.

Once the model is trained, the estimation of the next chord after a specific chord progression can be done giving to the RNN the single chords as input and considering only the output after the last chord of the sequence. The sequential structure of the model will ensure that the probability of the last chord will depend on all the chords given as input $p(x_i|x_{i-1}, ..., x_0)$.

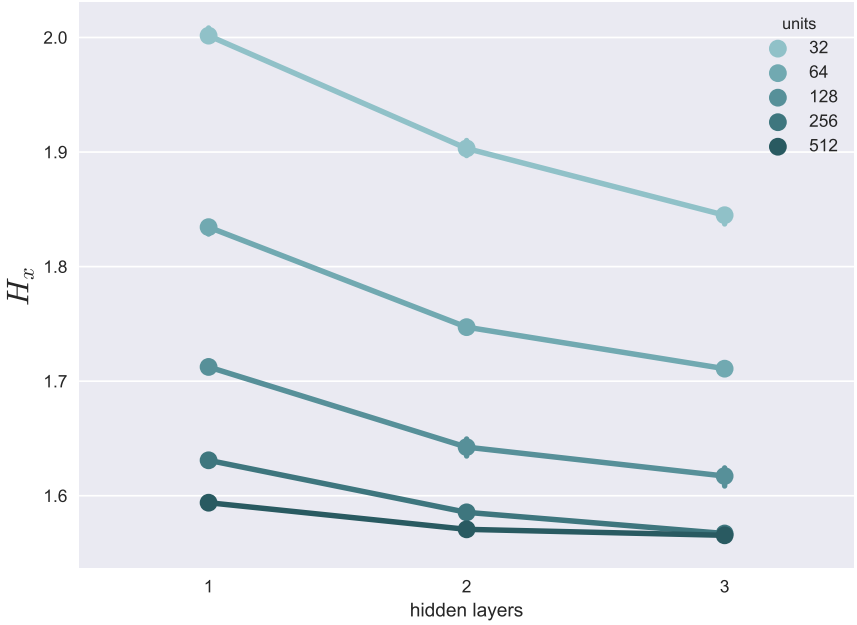

Figure 3.11: *The cross-entropy of the model, varying the number of the units and the depth of the hidden layer [Taken from [17]]*

.

3.3.4 Compound Model

We saw that the three proposed models have different advantages and disadvantages. In order to include as much good features as possible from each model, Di Giorgi [17] created a compound model, i.e. a weighted average of the three models:

$$p(x_i|x_{i-1}, ..., x_0) = \sum_{m \in M} \pi_m p_m(x_i|x_{i-1}, ..., x_0) \tag{3.23}$$

where π_m is the weight parameter for each model m and $\sum_{m \in M} \pi_m = 1$.

The set M is composed by the best performing models for each model type (PPM, HMM and RNN). In particular, the following settings were used:

- model C with backoff smoothing for PPM;
- $|Z| = 1000$ for HMM;
- the network with 3 layers of 128 units for RNN.

In order to find the best values for π_m, he performed a grid search aimed at the minimization of the cross-entropy. The results of this computation point the lower cross-entropy ($2.33\ bit/sample$) in the average ($0.5/0.5$) between the PPM and the RNN models. This means that the information added from the HMM, given the PPM and the RNN is equal to zero.

CHAPTER 4

Experimental Results

In this chapter we present the design and the results of the listening test. The goal is to give an answer to the question: "Can we predict which level of harmonic complexity will suit best the taste of a specific user?" Our hypothesis is that this value could be related to the level of music expertise of the user. We know from the work of Di Giorgi [17] that the perceived complexity of a chord progression has a strong correlation with the probability of the sequence. So we will use the terms "more complex progressions" and "easier progressions" to identify respectively more and less probable chord sequences.

4.1 Setup of the Listening Test

We developed an experiment that is organized in two sections. In the first section we profile the level of "musical expertise" of the subject through a survey (Section 4.1.1). The second section is subdivided in 8 small tests where we consider the harmonic progressions produced by the different generative models presented in Chapter 3 and we ask the subject to select the harmonic complexity that he likes most. In order to reduce the cognitive load, we designed a particular interface which allows people of diverse musical expertise to make this choice and provides the result in a format that could be easily analyzed. The subject is provided with a big horizontal slider that he can move in order to generate a progression with a specific probability (details in Section 4.1.2). When dragging it to the left, the subject will listen to simpler progressions, whereas by dragging it on the right, more complex progressions are played. The subject can keep

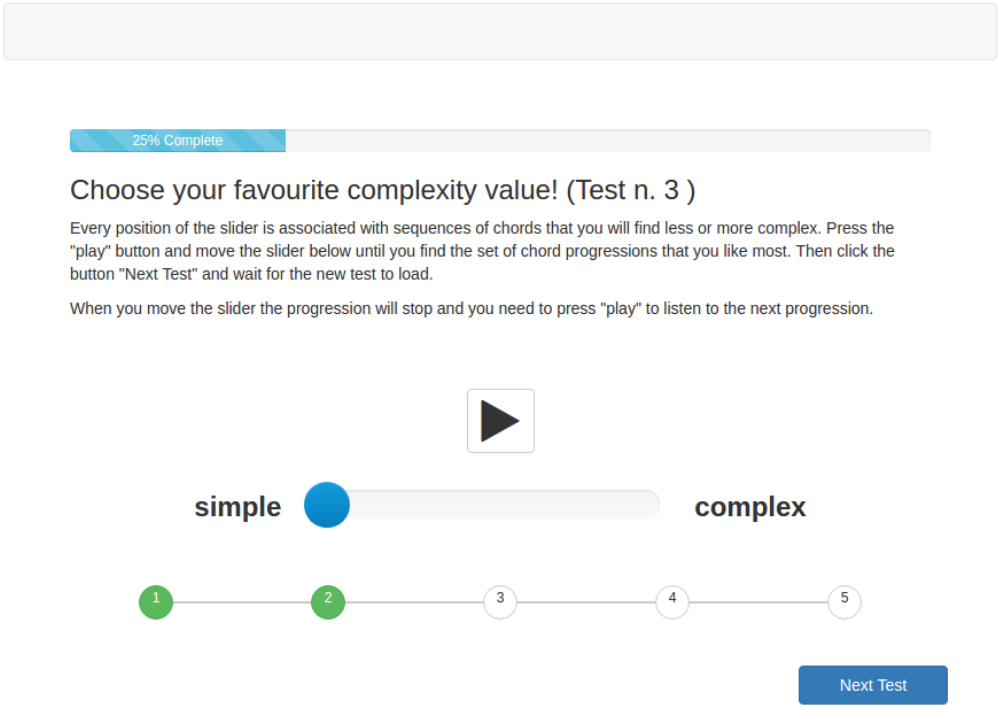

Figure 4.1: *The user interface (UI) for the selection of the preferred complexity.*

listening to chord sequences, moving the slider until he finds the complexity value that he prefers.

As said before, this test is repeated 8 times, in order to evaluate the results obtained with the different models (Compound, HMM, RNN, PPM) and different music modes (major, minor).

We ask the user to focus on the complexity of the current progression, but there is the risk that previous progressions would still affect the perception of the current progression. To minimize this risk, we used the following precautions:

- displaying a step graph with 5 circles, corresponding to the 5 chords of the sequences, to make evident which chord of the progression is currently played and when a new progression starts;
- inserting a pause of 1 second between progressions;
- separating the minor and the major modes in different tests.

4.1.1 Subject Profiling

Musical skills and expertise may vary a lot among different people. Also the concept of music expertise is a really wide concept, ranging from performance of an instrument and listening expertise, to the ability to employ music in functional settings or to communicate about music. We chose to use the *Goldsmiths Musical Sophistication Index*

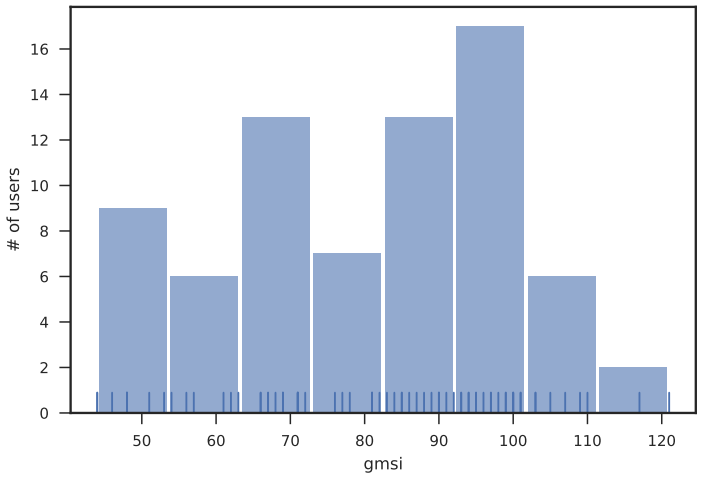

Figure 4.2: *The number of users that participated in the test, grouped by musical expertise using the General Musical Sophistication Index (GMSI).*

(GMSI) [48] because it can give an evaluation of "musical skills" for both musicians and non-musicians. The computation of the GMSI is performed through a survey with 38 questions and 7 answers for each question. The answers are then combined into 5 sub-factors (active engagement, perceptual abilities, musical training, singing abilities, emotions) and one general factor (general music sophistication).

The GMSI distribution of the people that participated to the test is shown in Figure 4.2.

4.1.2 Chord Progressions Generation

In this section we explain how we generated the chord progressions from our model. As stated before, sampling chord sequences with different complexity is equivalent to sampling chord sequences with different probability. In this task we use the capability of our models to output the probability of having a particular chord, given the previous chords (i.e. the probability of observing a particular chord transition). Di Giorgi [17] identified three techniques that can be used for sampling:

1. Range Sampling, i.e. dividing the probability interval $[0, 1]$ in K non overlapping sub-intervals $T_k = [\alpha_k, \beta_k)$ and sampling every chord x_i of the sequence from a subset of the alphabet $\hat{X}_k$ containing chords with a probability (given the context) that belongs to the interval T_k:

$$\hat{X}_k(x_{i-1}, ..., x_0) = \{x | p(x | x_i - 1, ..., x_0) \in T_k\} \tag{4.1}$$

   The complexity can then be controlled by choosing the interval k. A lower value of k generates less probable chords while a bigger value of k generates more probable chords.

2. Uniform Sampling, i.e. sampling each chord of the progression from an uniform

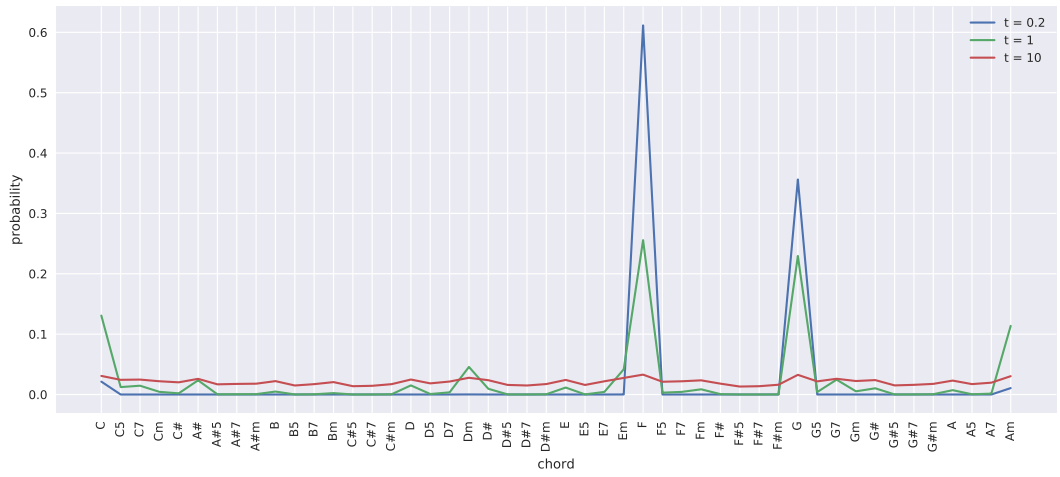

Figure 4.3: *The evolution of the temperature-modified probability mass function, changing the temperature parameter τ. In particular this is the probability after the first major tonic ($Cmaj$). With a temperature $\tau < 1$, the model will probably generate the chord F) or G, while for an higher value of temperature many chords can be generated.*

distribution over the set of chords X and then dividing the generated progressions in different bins, according to the progression probability.

3. Temperature Sampling, i.e. modifying the probability of the next chord (given the context) $p(x|x_i - 1, ..., x_0)$ with a *Temperature Function* and then sampling each chord of the sequence from the modified distribution. We choose as temperature function the following:

$$f(x) \mapsto \frac{f(x)^{1/\tau}}{\sum_{x \in X} f(x)^{1/\tau}} \tag{4.2}$$

The purpose of the temperature function is to flatten or unflatten the probability distribution. We can better understand it by considering some values of the parameter τ. For $\tau = 1$, the temperature function is just the identity function so the distribution is unaltered. For $\tau \to 0$ the temperature function moves all the probability of the distribution on the most likely outputs, turning the sampling process into the argmax function. For $\tau \to \infty$ the the distribution tends to become uniform over all the symbols (Figure 4.3).

We decided to use a combination between temperature sampling and uniform sampling. Uniform sampling allows to have the sequences split into non-overlapping bins, while sampling from the temperature modified distribution allows to produce more easily chord progressions of all possible probabilities. Range sampling was instead discarded because a progression with the same transition probability between each chord is not what we usually find in music. In fact, basic composition rules suggest to mix complex chord transitions with simpler chord transitions in order to keep the harmony understandable. We assume that making a sequence of complex transition without an harmonic release, would instead make the user forget the harmonic context; and without an harmonic context all the perceptual effects of tonal harmony would not be

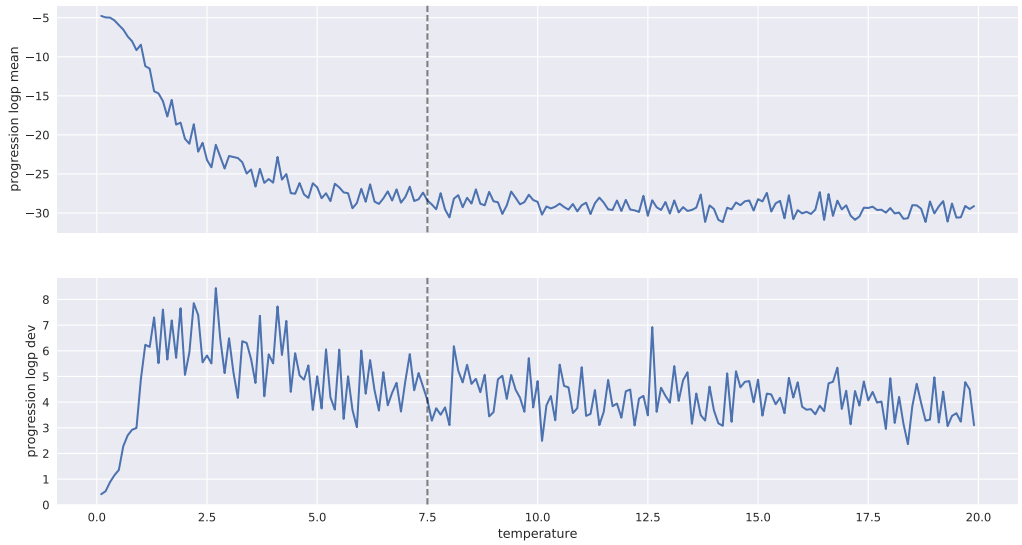

Figure 4.4: *Mean and variance of the log-probability of a set of 5 progressions generated with different temperature value.*

perceived.

In order to understand which temperature to use for our sampling, we proceed with a simple heuristic where we evaluate the mean and variance of a group of 5 progressions generated with a certain temperature τ. The temperature parameter was changed from 0 to 20, with a step of 0.1. From the results (Figure 4.4) we can observe that $\tau = 7.5$ can be a good upper bound for the temperature parameter, since progressions generated with an higher temperature are likely to have the same probability.

In order to avoid another possible source of bias in the results, we decided to force the last chord of the progression to be the tonic. This is due to the fact that the ending chord of a progression could influence its perceived complexity in a way that is not considered by our model. For example, if we compare the progressions $C - Dmin - G - C$ and $C - Dmin - G$, their (normalized) log-probability (computed with the compound model) is very similar (~ -7.3). But from a perceptive perspective, the first progression will sound more "stable" and "conclusive" than the second [53], and that could influence an user evaluation of perceived complexity. Since in tonal harmony we expect a song to start and end with the tonic, forcing it at the end had the side effect of making the progressions sound more realistic. In particular, with this method we reduce the possibility that a progression with a low probability would be perceived as "random". We have already discusses in Section 1.1 that for random stimuli the perceived complexity level can be difficult to evaluate. This is what we want to avoid, otherwise low probability sequences would be difficult to compare by people. Since all the language models that we use consider the generation of the next symbol depending only on the past symbols, it is not easy to constrain the ending symbol analytically. So we decided to adopt a "brute-force" by producing a big set of chord progressions and

by filtering out those that were not ending with the tonic chord.

Table 4.1: *A subset of the progressions generated by the Compound model, grouped in different bins, depending on their log probability.*

Major Progressions		
bin	log probability	progression
1	-5.667693	C Am F G C
	-6.460180	C G F G C
2	-7.064298	C G C G C
	-6.650036	C G C F C
3	-8.380957	C Dm C Dm C
	-8.129084	C F Dm G C
5	-10.431535	C E7 Am F C
	-10.087513	C D C D C
10	-16.131764	C B A# A C
	-15.890421	C D#m Dm G7 C
20	-26.125212	C A7 A#7 D# C
	-26.681781	C G F#7 A C

Minor Progressions		
bin	log probability	progression
1	-7.026709	Cm Fm A# D# Cm
	-6.826959	C G F G C
2	-8.208561	Cm G# Fm A# Cm
	-8.432002	C G C F C
3	-8.801180	Cm Fm G# A# Cm
	-9.296664	C F Dm G C
5	-10.956227	Cm G Fm G Cm
	-11.122792	C D C D C
10	-16.763926	Cm Dm G# Dm Cm
	-16.659720	C D#m Dm G7 C
20	-27.258201	Cm C#m G7 G# Cm
	-27.445176	C G F#7 A C

After the temperature sampling (and the deletion of the progressions not ending with tonic), we computed the probability of each progression and we distributed them over 30 non overlapping probability bins. In the Table 4.1 a subset of the generated progression is shown. The classification of the chord sequences in bins of different probability performed by our model reflects quite well the complexity that would have been assigned using the rules of music theory. For example, considering the major progressions, we notice that the progressions inside bins 1 and 2 contains only I, IV, V, VI; in the bin 3 is introduced the chord II; in the bin 5 we can find a secondary dominant of the VI; in the bin 10 are introduced out-of-key chords, that resolve chromatically on chords of the tonality.

Voicings. Since we need to obtain audio excerpts from our progressions, we need to deal with the choice of the voicings. This is a factor that greatly influences the perception of a harmonic progression. The easiest example is to listen to chords played only in the fundamental position. The absence of a melodic interconnection between the chord notes will make every progression sound "artificial" and not enjoyable. Therefore, we use the simple voice leading model presented by Di Giorgi [17], in order to obtain a more "musical" result, without characterizing too much the harmonic progressions with strong melodic lines that could alter the perception of the complexity. Every progression generated by the model was thus processed by this "voice leading model" in order to obtain a list of voicings from a list of chords, where each voicing is described by a list of MIDI note numbers.

4.1.3 Web App

To make the test easily reachable by many people, we decided to implement it as a web app. As hosting service, we chose Heroku, a cloud *platform as a service* (PaaS) supporting several languages. Using a Service-Oriented Architecture is really convenient

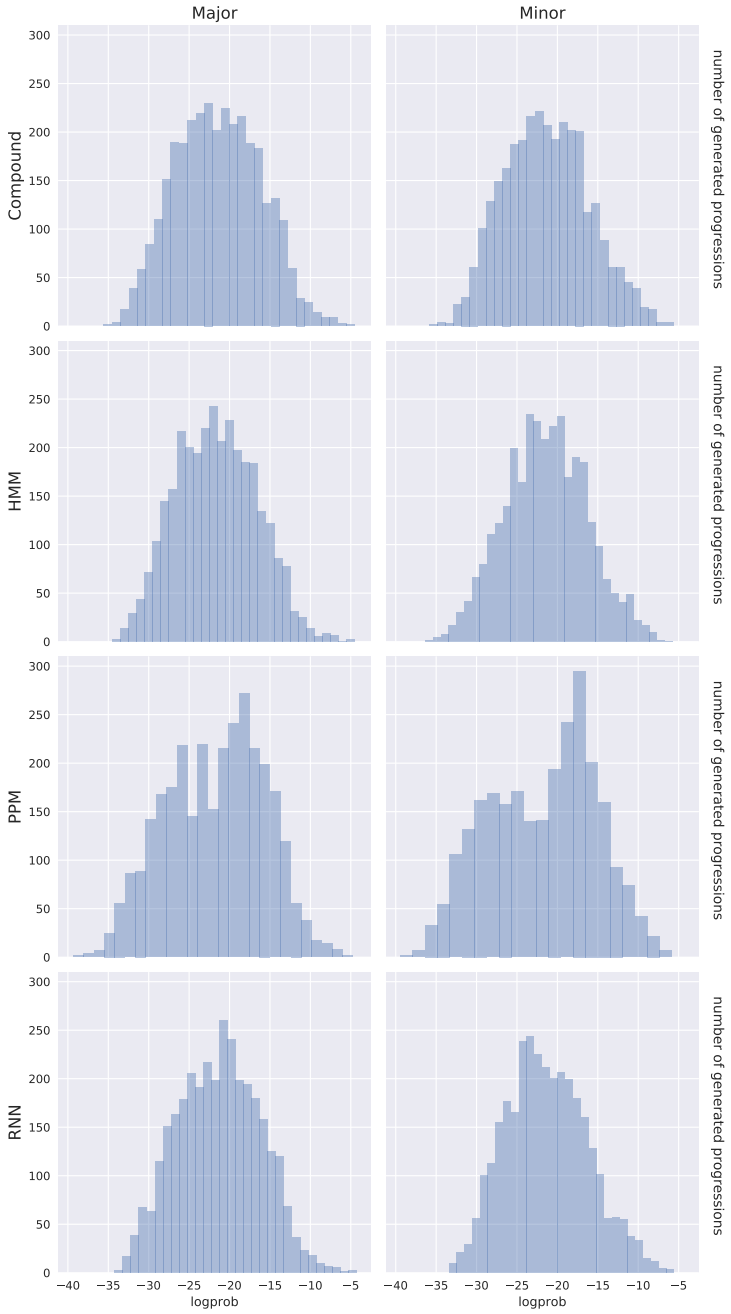

Figure 4.5: *The number of generated progressions that end with tonic for each model, distributed in 30 non overlapping bins according to their log probability*

for this kind of "lightweight" applications, since it allows the programmer to easily deploy on the web a local app with a simple `git` commit, without working on a server and database configuration. The free plan of Heroku was perfectly suited for the technical specification of this app.

Server side architecture

The server side architecture was developed in NodeJS, an open source server framework that allows the use of JavaScript on the server. We chose it because it enables non-blocking, asynchronous programming, which is very resource efficient; a single thread was enough to allow multiple people to execute the test simultaneously.

The Server Side of this application is just a set of REST API. In particular the exposed endpoints are:

- `GET /api/auth` to have the unique user id.
- `GET /api/:table` to have the set of progressions (major or minor) generated by a specific model (Compound, NGram, HMM, RNN).
- `POST /api/result` to save the survey answers and the choosen complexities.
- `POST /api/comment` to save the email and comment of the user.

There is also another set of endpoints that is used for the result analysis in order to retrieve the experimental results from the server.

To store the progressions and the results and to perform conflict resolution in case of simultaneously access to them, it was used a *PostgreSQL*, an open source database. In particular we used Heroku Postgres, a SQL Database-as-a-Service. We stored for each chord of our progressions, the array of note-numbers that was played during the test. For the entire listening test, the content of the database can be summarized as:

- 4 generation models (compound, HMM, RNN, n-Gram).
- 2 modes for each model (major and minor).
- 30 probability bins for each mode. We chose this number to give the user the feeling of a continuous slider, to obtain higher resolution data for the regression.
- 10 progression (at most) for each probability bin. We chose to put multiple progressions in each bin not to make the user focused on a single chord progression, but rather on the complexity of a set of chord progressions. Anyway we capped this number at 10, since the user would never listen to more progressions before changing bin.
- 5 chords for each progression (the last chord and the first are the tonic, but are saved separately because they can have different voicings).
- 5 notes for each chord.

Client side architecture

The client-side was developed using Knockout, a JavaScript library that makes it easier to create rich and interactive interfaces. It is based on the MVVM (Model View View-Model) architectural pattern that facilitates the writing of a clean and reusable code. From a practical point of view, most of the user interactions in our web app are handled locally, avoiding resource wasting calls to the server. The server API are only called to obtain the set of progressions and to send back the results of the experiment and the user's comments.

The structure of the client-side of our web app was designed using IFML (Interaction Flow Modeling Language), a modeling language made for expressing the content, user interaction and control behaviour of the front-end of software applications. In particular an IFML model allows developers to specify the following aspects of an interactive application [4]:

- The view structure and content: the general organization of the interface is expressed in terms of ViewElements, along with their containment relationships, visibility, and activation. Two classes of ViewElements exist: ViewContainers, i.e., elements for representing the nested structure of the interface, and ViewComponents, i.e., elements for content display and data entry. ViewComponents that display content have a ContentBinding, which expresses the link to the data source.
- The events: the occurrences that affect the state of the user interface, which can be produced by the users interaction, the application, or an external system.
- The event transitions: the consequences of an event on the user interface, which can be the change of the ViewContainer, the update of the content on display, the triggering of an action, or a mix of these effects. Actions are represented as black boxes.
- The parameter binding: the input-output dependencies between ViewElements and Actions

The IFML model of our application (Figure 4.6) is is quite simple, since the user is supposed to navigate in just one direction through the test. There is just one loop in the `Test` View Model, in order to perform the test with the different progression sets reusing the same code.

To implement the IFML model, we used IFMLEdit[1] a web-based open source tool developed by Bernaschina [4]. This tool allows the specification of IFML models, the investigation of their properties with an easy to access web representation, and the generation of code for web and mobile architecture.

To play the note numbers we used the package `soundfont-player`, a soundfont loader/player to play MIDI.js sounds using WebAudio API. The sound generation part is completely up to the client. This is a lightweight solution that allows to avoid the downloading of heavy audio-files from the server.

[1] http://info.ifmledit.org/

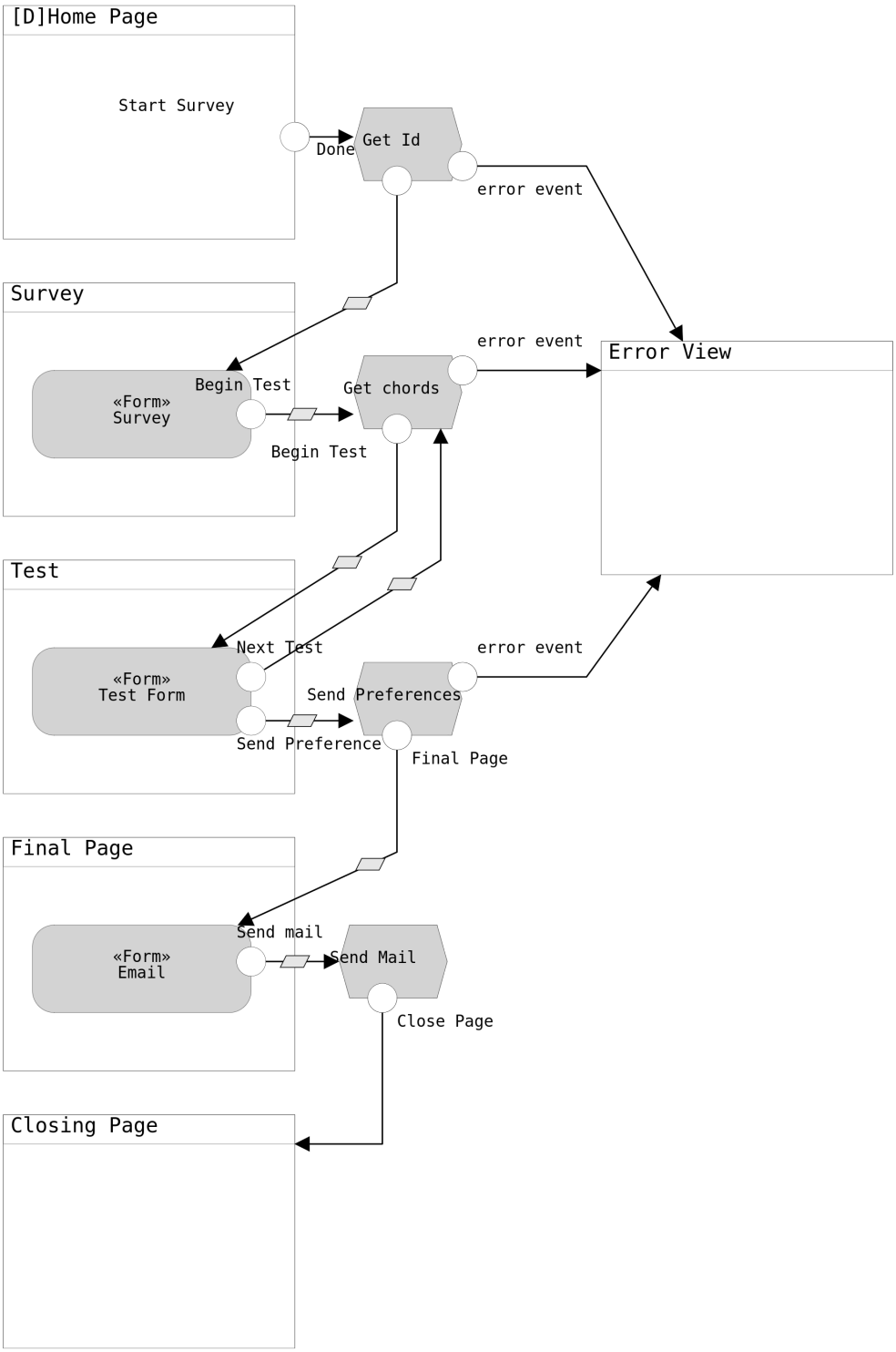

Figure 4.6: *The IFML model of the client side of the Web App developed for the test.*

4.2 Result Analysis

In this section we analyze the results of the listening test in order to evaluate if there is any correlation between the music sophistication of subjects and the degree of complexity that they prefer.

In Figure 4.7 we can see the distribution of the choices of the different people. Every single point represents the complexity level (measured with the log probability of the sequence) chosen by the subject.

The kind of knowledge that we are trying to extract from this test can be classified by using a classification proposed by Law et al. [37], as *cultural truth*, i.e. some sort of perceptual judgment refers to the shared beliefs amongst the set of people that we sample. Examples include mapping a piece of music to the emotions that it evokes, determining whether a particular website contains pornographic content or not, and rating the attractiveness of a celebrity. In these cases, it is difficult to derive an objective, true answer. This is because "serenity", "pornography" and "attractiveness" are perceptual concepts that are difficult to define precisely, and their interpretation can vary greatly from individuals to individuals. We rely on the *wisdom of the crowd* to establish what is truth, i.e. we assume that there is a cultural consensus among the subjects that can be identified even if some workers may deviate from the norm.

The development of the test as a public web application allowed to collect data very fast, but it allows the occurring of some possible "bad situations" which produce noise in the results. In the following list we enumerate these possible problems, along with the solution that we adopted in order to minimize their occurrence or mitigate their effect.

1. People not understanding exactly the test instructions. To reduce the risk of this problem, we carefully design the user interface, in order to make it very intuitive. For the same reason, we decided to ask a single task to the user.
2. People failing the test on purpose: the separation between the subject and the test manager can reduce the social responsibility from the subject, that can behave in unpredictable ways without any reason in particular. We tried to avoid this problem by proposing the test only to people that could relate to us in some way. This included friends on social network, private messages and mailing list of field-related people (e.g. the International Society for Music Information Retrieval - ISMIR mailing list). Moreover, the user was encouraged to write his email or some comments on the test in the final page, with the effect of increasing his perception of responsibility on the result of the test. Advertising the test on public pages of big websites would have dramatically increased the pool of participants, but the risk of this kind of "bad" behavior was too high.
3. People getting tired of the test and completing it with random answers. We tried to minimize this behavior by using a game-like interface, however the incidence of this behavior may still be quite high. In the following section we propose a heuristic method to discriminate between "meaningful" and "non-meaningful" results.

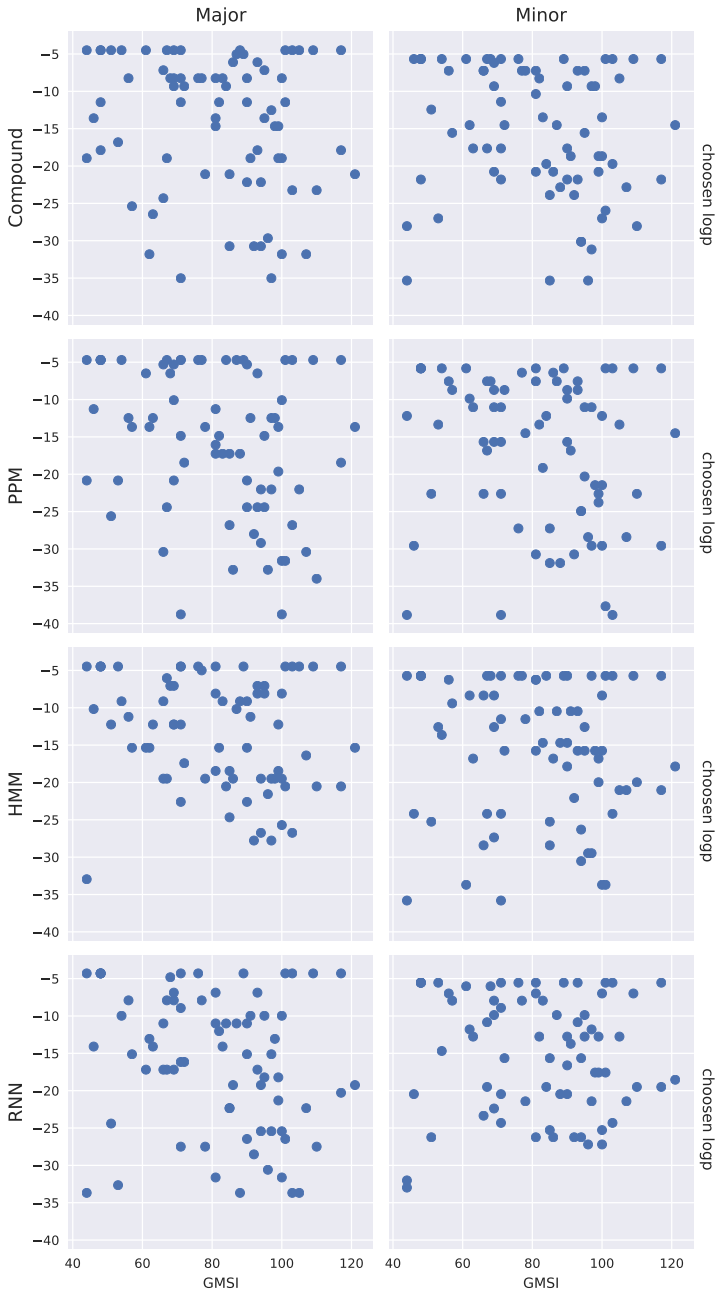

Figure 4.7: *Graphical representation of the "raw" results of the eight different tests. Each plot represent a single test and every point is the complexity chosen by a subject with a certain GMSI.*

We have a big amount of variables to analyze, since every subject performed 8 tests and each test output one final level of complexity. Moreover we choose to save also all the progressions that the user heard, before making the final decision. Let us introduce some notation to make the analysis easier to understand:

- N is the total number of subjects that participated in the experiment.
- a_i^n is the single final answer of the user n for the test i. To have an intuitive notation we use $i \in \{$'CM', 'Cm', 'PM', 'Pm, 'HM', 'Hm', 'RM', 'Rm'$\}$ where, for example, Cm indicates the compound model with minor tonic and RM indicates the RNN model with major tonic. This value can be measured with the complexity bin number that the user chooses (from 0 to 29 in an increasing level of complexity) or with the log probability of the bin chosen. These two measures are completely equivalent for the experiments and we will use one or another depending on what is convenient for the situation.
- $\vec{a}_n^i$ is the vector of the progressions listened by the user n for the test i. The last item of this vector, corresponds to a_i^n.
- $t_i = \{a_i^1, a_i^2, ..., a_i^N\}$ indicates the vector of final results for the test i.
- $e_n = \{a_{CM}^n, a_{Cm}^n, ..., a_{Rm}^n\}$ indicates the vector of the final answers of the user n for all the tests.
- adding the tilde on top of all the previous definitions $\tilde{x}$, means that the value is "valid" or that the vector is composed only by "valid" values. We consider as "valid" the values that are not discarded by the cleaning function defined in Section 4.2.1.

4.2.1 Data Cleaning

In order to detect non-meaningful selections, we propose a heuristic method that exploits our knowledge of the process of "preferred complexity selection". If the user could glean information only from the listening of the audio, he would have to listen every single bin of complexity before doing his choice (not feasible). However we are providing another piece of information: that the complexity will increase monotonically moving the slider from left to right. We thus expect the user to perform a sort of bisection method to find the best complexity. In other words, we do not expect a subject to truthfully choose the preferred complexity, if he did not listen to at least one more complex progression and one simpler progression (Figure 4.8). An easy algorithm that implements this idea is: looking at the discrete derivate of the "listening path" $\triangle(\vec{a}_n^j)$ and considering the result valid only if it is positive and negative at least once.

We can see from the Figure 4.9 that we removed a lot of non-meaningful points, such as all the minimum values generated by people listening to just the first progression and proceeding to the next test. On the other side, this method reduced drastically the useful data-points for the analysis (Figure 4.10). We can notice from Figure 4.11 that the number of meaningful results tend to decrease, when the number of test already done by the user increases. This can be explained by the user getting tired of the test, predicting what he will like and listening only to one final progression.

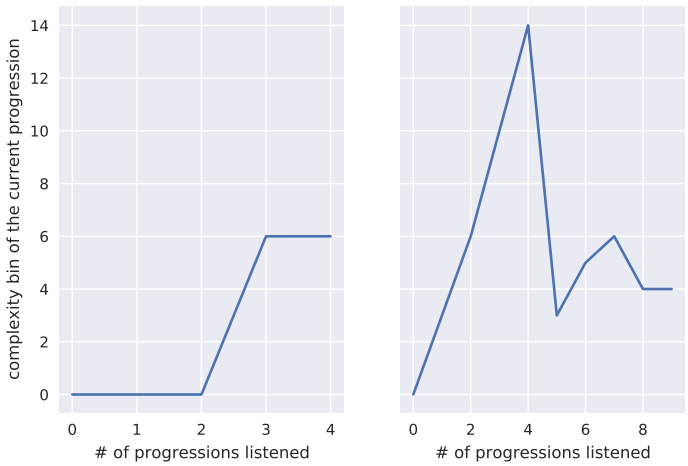

Figure 4.8: *Example of a meaningful pattern of choice (left) and non-meaningful pattern of choice (right).*

Table 4.2: *Pearson and Spearman correlation between each test results and the user GMSI.*

	Entire Dataset				Cleaned Dataset			
	Pearson	Pear p-value	Spearman	Spear p-value	Pearson	Pear p-value	Spearman	Spear p-value
Comp. (maj)	-0.181473	0.121768	-0.178210	0.128734	-0.120718	0.398770	-0.141527	0.321859
Comp. (min)	-0.172435	0.141800	-0.224383	0.054613	-0.231388	0.121805	-0.312348	0.034572
PPM (maj)	-0.299217	0.009604	-0.309269	0.007336	-0.314285	0.037739	-0.374572	0.012244
PPM (min)	-0.210335	0.072059	-0.231151	0.047534	-0.279783	0.080378	-0.314933	0.047778
HMM (maj)	-0.206847	0.079115	-0.218115	0.063768	-0.456597	0.007563	-0.506102	0.002656
HMM (min)	-0.113838	0.334175	-0.166785	0.155526	-0.330472	0.049004	-0.398774	0.015992
RNN (maj)	-0.248327	0.032895	-0.272805	0.018693	-0.462175	0.008855	-0.522799	0.002549
RNN (min)	-0.083592	0.481986	-0.129051	0.276533	0.221028	0.232124	0.156595	0.400200

By examining the correlation between GMSI and the log probability t_n (Table:4.2) we can generally confirm the existence of a relationship between the user music expertise and the complexity that he prefers. We can notice that, generally, the cleaned dataset $\tilde{t}_n$ gives higher correlation values. The first test (Compound maj) and the last test (RNN min), instead, behave in an unpredicted way, but this can be explained by the noise sources described in the previous section. We can also notice that Spearman correlation performs always better than Pearson correlation. This could mean that the correlation between the two variables is monotonic, but not linear (Section 2.2.1). Therefore we try to fit the data with different polynomial regression models.

4.2.2 Polynomial Regression

We choose to fit the results with 5 different polynomial models, from degree 1 to degree 5. In order to mitigate the risk of overfitting, we evaluate every polynomial model using 300 steps of cross-validation where at each step the 80% of the dataset is used for the training while the remaining 20% is used for testing. As metrics to evaluate the quality of the fitting, we used R^2, mean squared error, explained variance score and median absolute error. We notice that, for the compound model with minor chords, (Figure 4.12)

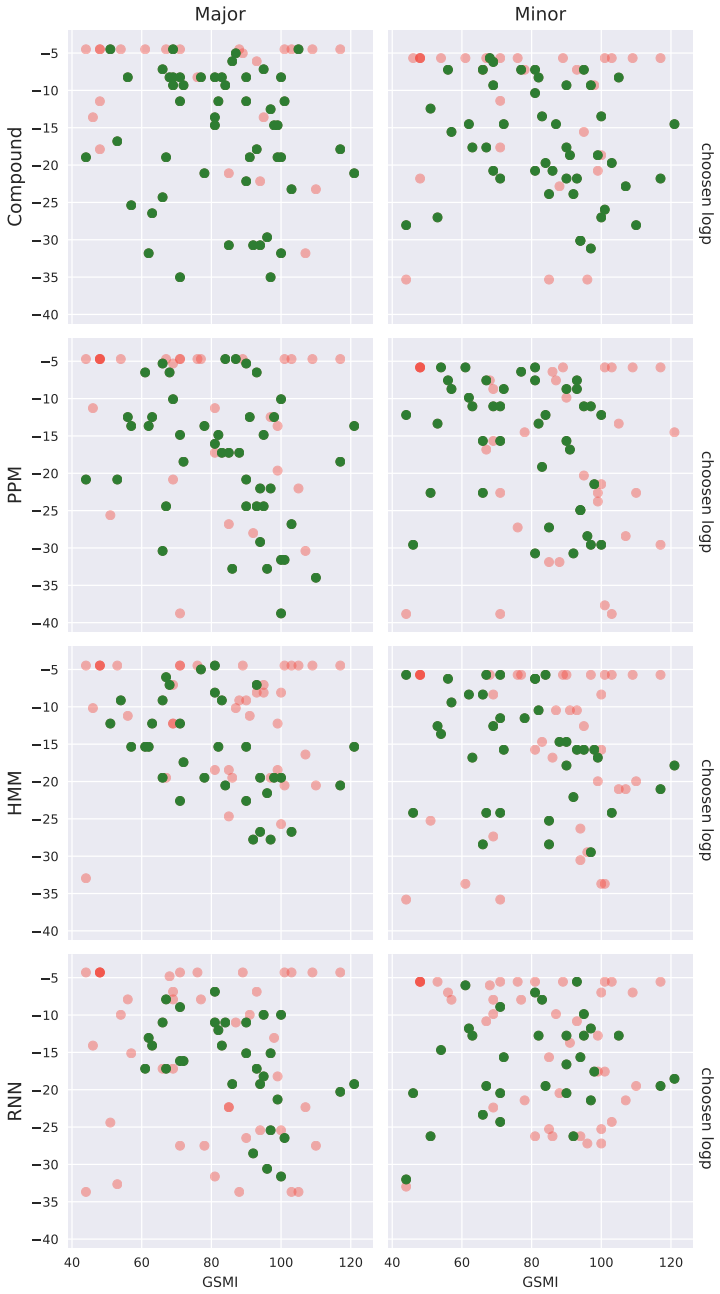

Figure 4.9: *Graphical representation of the "cleaned" results of the eight different tests. Each plot represent a single test and every point is the complexity chosen by a subject with a certain GMSI. The green color marks the accepted point, while the red color marks the non-significant points, according with the cleaning function that we defined.*

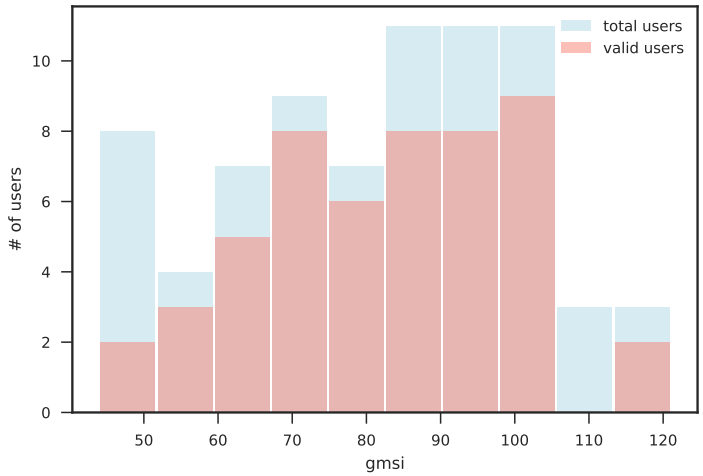

Figure 4.10: *Number of meaningful results against users GMSI, for the test about the compound model with major tonic.*

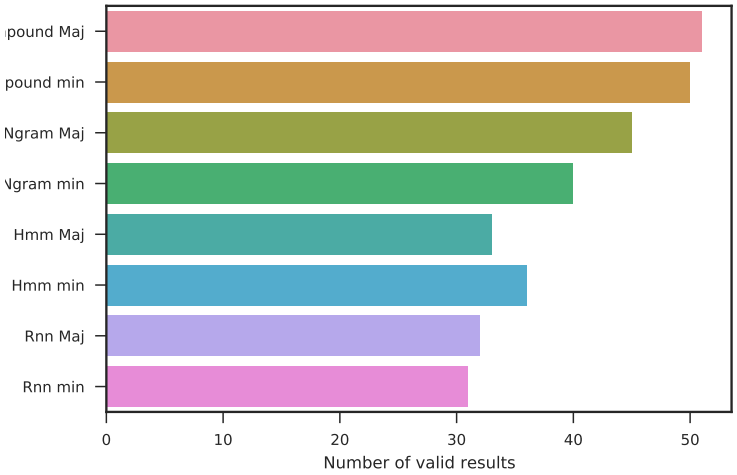

Figure 4.11: *Number of meaningful results for each test.*

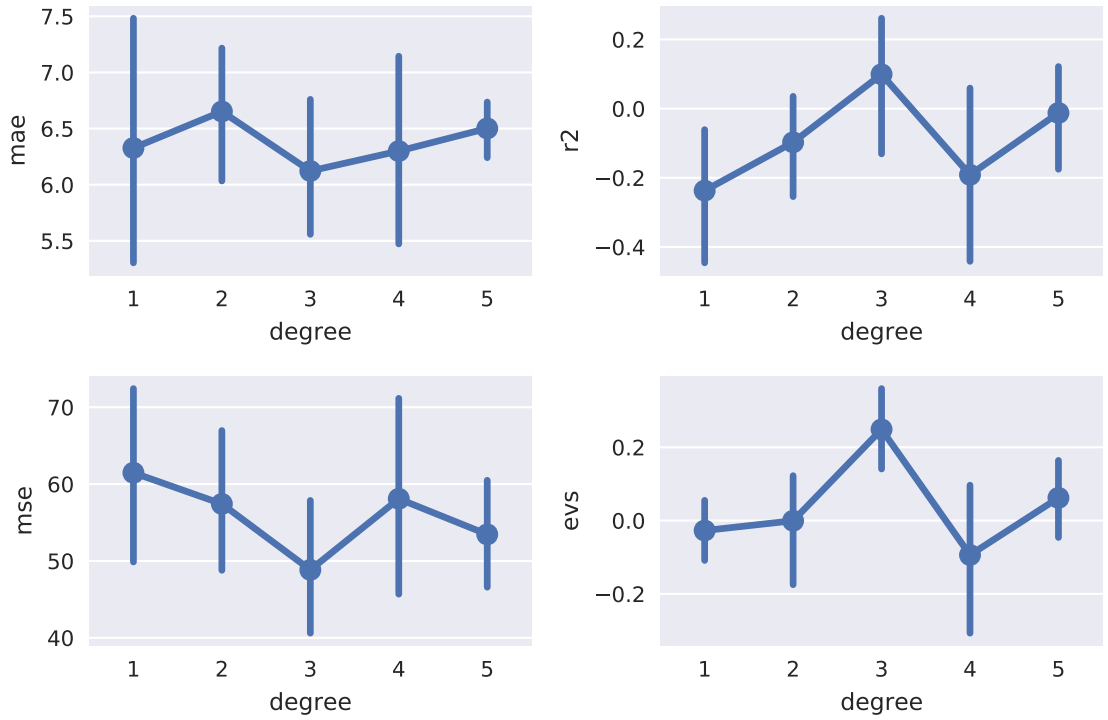

Figure 4.12: *Evaluation of polynomial regression with different metrics: R^2, mean squared error (MSE), explained variance score (EVS) and median absolute error(MAE).*

the model with the best performances is the 3rd order polynomial. However the R^2 values is always very low and often negative. This can be explained by considering that, due to the nature of this test, we have a high probability of having outliers in the results. As pointed out in [56] and [19], the Mean Squared Error (and thus the R^2) can behave badly when the error distribution is not normal, particularly when the errors are heavy-tailed.

4.2.3 Robust Regression

In order to mitigate the effect of the outliers during the regression, we decided to use the Huber estimator with $k = 345\sigma$ since it produces 95-percent efficiency when the errors are normal, and still offers protection against outliers. We train the Huber estimator for different polynomial models, similar to what we did in section 4.2.2, by using cross validation and computing Median Average Error (MAE) on the portion of data that was not using for training. We used the Median Average Error, since it is a metric that we can trust also in presence of outliers. The regression model with the best performances in term of Median Average Error is the 2nd degree polynomial. In Figure 4.13 we can see the results for the Compound model with minor chords t_{Cm}.

In the Figure 4.14 we can see the regression line generated by the Huber for each test t_i. All the tests behave similarly (except for the RNN with minor chords, that we already pointed out as having very noisy results).

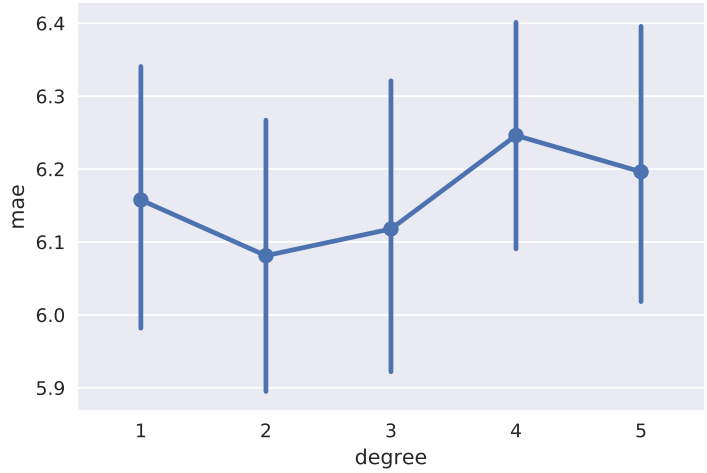

Figure 4.13: *The MAE of the regression using polynomial models of different degrees and the Huber estimator. The fitted data are the data generated by the Compound model with minor chords.*

4.2.4 Inter Model Statistics

We are also interested in comparing how every single subject considered the chords produced by the different generative models, i.e. to study the disposition of the values of the vector e_i.

We decided to use the same technique presented in Section 4.2.1 to know which data to discard. In particular we created two new subsets of our dataset, were each entry e_n is assigned under the following conditions:

- `all valid`: contains the results relative to the subjects that complete each one of the 8 test with valid results, according to the cleaning function, i.e. the collection of what we denoted previously as $\tilde{e}_n$. This dataset contains 8 entries and we denote it as $\tilde{E}$.
- `any valid`: contains the results relative to the subjects that completed at least one of the 8 test with a valid result. This dataset is a a superset of $\tilde{E}$ and contains 62 entries. We denote the entries as$\dot{e}_n$ and the dataset as $\dot{E}$.

The remaining 4 entries were completely discarded.

We computed the mean absolute deviation (MAD) of the chosen complexity bin for every entries e_n, obtaining a vector of MADs for each dataset. The choice of MAD instead of standard deviation was made for two reasons: having a measure less sensible to outliers, and having a measure with a more intuitive meaning. The analysis of these vectors reveals that people tend to be stable in the choice of their preferred complexity, moving the final result of 3.73 bins in average, along the different tests. This result is valid for both datasets (Figure 4.15), since the mean of these vector is almost the same: 3.74 for $\tilde{E}$ and 3.73 for $\dot{E}$. However, $\tilde{E}$ achieved better results since the standard variation of its vector is 1.20, against the 1.88 of the vector relative to $\dot{E}$. Taking off the first test t_{CM} from the statistic improves the mean of the vector related to $\tilde{E}$ by 0.2,

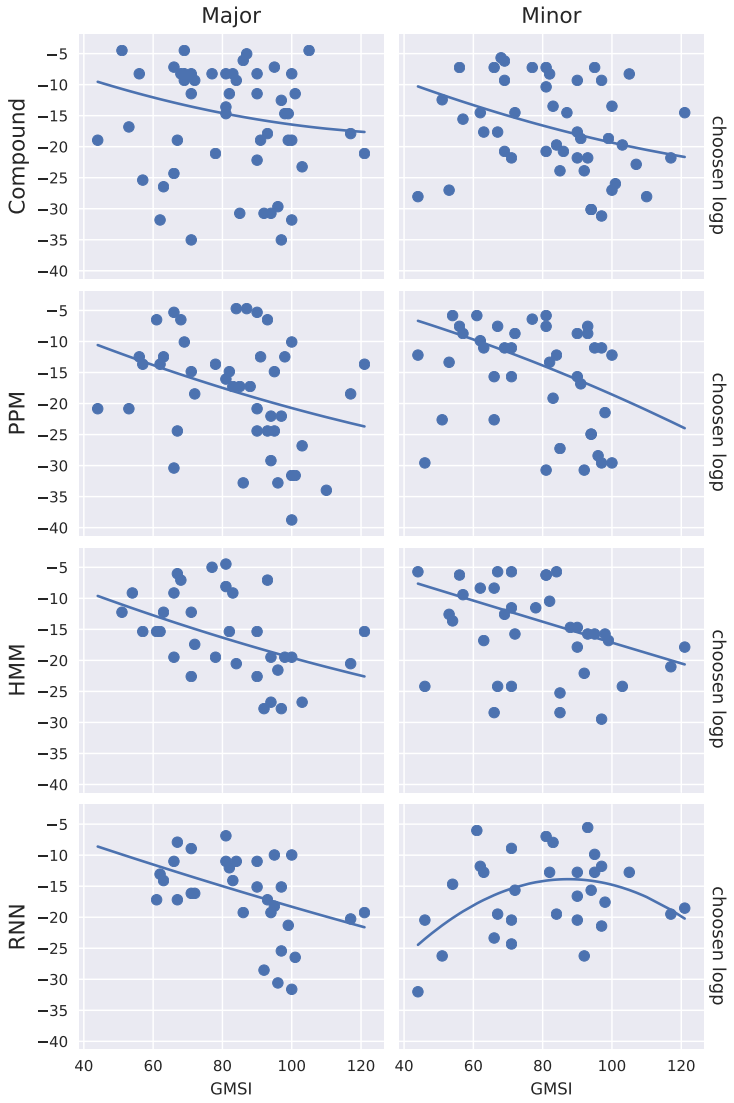

Figure 4.14: *The 2nd degree polynomial Huber regression line plotted for each test.*

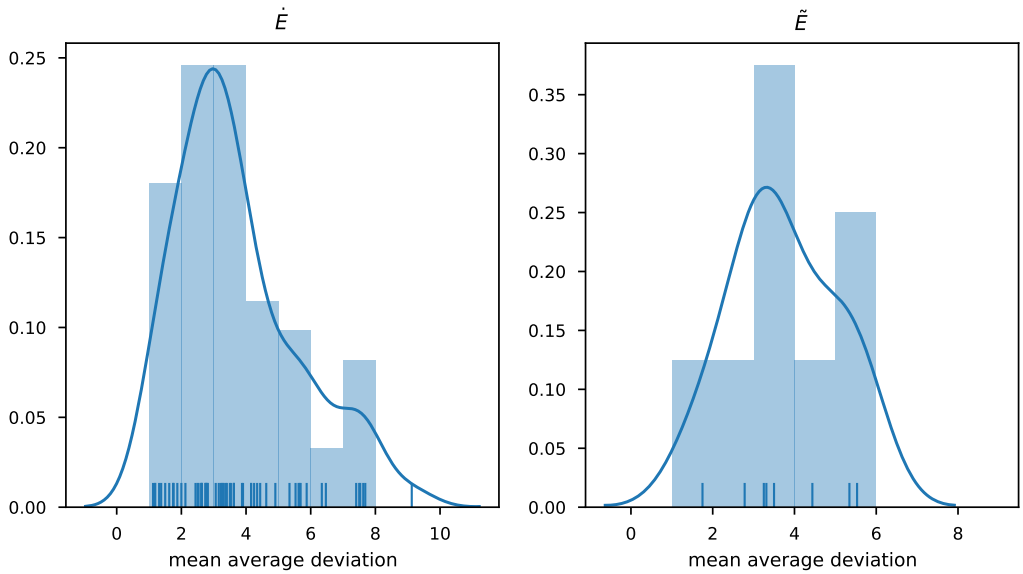

Figure 4.15: *The mean absolute deviation in the choice of the different subjects for the test ($MAD(\dot{E})$) on the left and ($MAD(\tilde{E})$ on the right).*

and the one related to $\dot{E}$ by 0.1. Instead, by ignoring the results of the last test t_{Rm}, we do not observe any significant change. This could confirm the idea that the unique behavior of $t_C M$ in the fitting task was due to people not properly understanding the instructions, while the unique behavior of $t_R m$ was instead given by people getting tired of the test and predicting their favorite progression depending on the previous tests, without listening to the needing number of progressions.

No correlation where found in both datasets between the variability of the selections and the GMSI of the subjects, since the p-value of the correlation between these two vectors gave a p-value > 0.7.

CHAPTER 5

Conclusion

In this thesis we analyzed the relation between the music expertise of a person and the level of complexity that he or she prefers for chord sequences. We used as a measure of music expertise the General Music Sophistication Index (GMSI) and as a measure of the complexity, the probability of the sequence of chords generated by a machine learning language model.

A special attention was given to the chord sequences generation, by forcing the first and the last chords to be the tonic, in order to enforce the Tonal behavior and avoid the generation of sequences that would have been perceived as random sequences. This guaranteed a good level of discernment also between sequences with a low probability.

In order to prove the correlation, we designed a listening test were the subjects had the possibility of intuitively choosing the progression with the preferred level of complexity, among other progressions with different complexities. We developed this test as a web application in order to reach a sufficient number of subjects for the test.

Due to the high level of freedom left to the subject during the test and the consequent high probability of having outliers, we decided to analyze the data with robust regression techniques. The results, by using an Huber estimator, confirm the existence of a positive correlation between the subject music expertise and the complexity of the progression chosen. That means, in other words, that people with an higher music expertise tend to prefer more complex chords.

We also detected no difference between the user preference for chord sequences gen-

erated with different machine learning models. That means that, from a perceptive point of view, all the models that we tested generate similar chord sequences with a comparable complexity.

5.0.1 Future Work

We obtained good results about chords sequences, however this is just a minimal part of the elements that compose a musical piece. For the future it would be interesting to repeat a similar procedure, by working with other important musical aspects such as rhythm, melody, chord voicings and orchestration. This would allow to arrive to a global model of prediction of the user preference that would not only greatly improve MIR classification and recommendation tasks, but would also give useful insight in the field of music composition and musical analysis.

Bibliography

[1] R Alpert. Perceptual determinants of affect. *Unpublished master's thesis, Wesleyan University*, page 51, 1953.

[2] Daniel E Berlyne. Novelty, complexity, and hedonic value. *Attention, Perception, & Psychophysics*, 8(5):279–286, 1970.

[3] Daniel E Berlyne. *Aesthetics and psychobiology*, volume 336. JSTOR, 1971.

[4] Carlo Bernaschina, Sara Comai, and Piero Fraternali. Ifmledit. org: model driven rapid prototyping of mobile apps. In *Proceedings of the 4th International Conference on Mobile Software Engineering and Systems*, pages 207–208. IEEE Press, 2017.

[5] Emmanuel Bigand and Marion Pineau. Global context effects on musical expectancy. *Attention, Perception, & Psychophysics*, 59(7):1098–1107, 1997.

[6] George David Birkhoff. *Aesthetic measure*, volume 9. Harvard University Press Cambridge, 1933.

[7] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[8] Michael J Burke and Mark C Gridley. Musical preferences as a function of stimulus complexity and listeners' sophistication. *Perceptual and Motor Skills*, 71(2):687–690, 1990.

[9] Michael A Casey, Remco Veltkamp, Masataka Goto, Marc Leman, Christophe Rhodes, and Malcolm Slaney. Content-based music information retrieval: Current directions and future challenges. *Proceedings of the IEEE*, 96(4):668–696, 2008.

[10] Stanley F Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, pages 310–318. Association for Computational Linguistics, 1996.

[11] John Cleary and Ian Witten. Data compression using adaptive coding and partial string matching. *IEEE transactions on Communications*, 32(4):396–402, 1984.

[12] John G Cleary and William J Teahan. Unbounded length contexts for ppm. *The Computer Journal*, 40(2_and_3):67–75, 1997.

[13] Pedro P Cruz-Alcázar and Enrique Vidal-Ruiz. Modeling musical style using grammatical inference techniques: a tool for classifying and generating melodies. In *Web Delivering of Music, 2003. 2003 WEDELMUSIC. Proceedings. Third International Conference on*, pages 77–84. IEEE, 2003.

[14] Pieter-Tjerk De Boer, Dirk P Kroese, Shie Mannor, and Reuven Y Rubinstein. A tutorial on the cross-entropy method. *Annals of operations research*, 134(1):19–67, 2005.

[15] W Bas De Haas, José Pedro Magalhães, and Frans Wiering. Improving audio chord transcription by exploiting harmonic and metric knowledge. In *ISMIR*, pages 295–300, 2012.

[16] WB De Haas. *Music information retrieval based on tonal harmony*. PhD thesis, Utrecht University, 2012.

[17] B. Di Giorgi, S. Dixon, M. Zanoni, and A. Sarti. A data-driven model of tonal chord sequence complexity. 2017.

[18] Bruce Edmonds. What is complexity?-the philosophy of complexity per se with application to some examples in evolution. In *The evolution of complexity*. Kluwer, Dordrecht, 1995.

[19] John Fox. Robust regression. *An R and S-Plus companion to applied regression*, 2002.

[20] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[21] Alex Graves et al. *Supervised sequence labelling with recurrent neural networks*, volume 385. Springer, 2012.

[22] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE, 2013.

[23] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[24] Jiang Guo. Backpropagation through time. *Unpubl. ms., Harbin Institute of Technology*, 2013.

[25] Pierre Hanna, Matthias Robine, and Thomas Rocher. An alignment based system for chord sequence retrieval. In *Proceedings of the 9th ACM/IEEE-CS joint conference on Digital libraries*, pages 101–104. ACM, 2009.

[26] Jan Hauke and Tomasz Kossowski. Comparison of values of pearson's and spearman's correlation coefficients on the same sets of data. *Quaestiones geographicae*, 30(2):87, 2011.

[27] Geoffrey E Hinton. Learning multiple layers of representation. *Trends in cognitive sciences*, 11(10):428–434, 2007.

[28] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[29] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[30] Paul Glor Howard. The design and analysis of efficient lossless data compression systems. 1993.

[31] Anssi Klapuri. Introduction to music transcription. *Signal Processing Methods for Music Transcription*, pages 3–20, 2006.

[32] Reinhard Kneser and Hermann Ney. Improved backing-off for m-gram language modeling. In *Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on*, volume 1, pages 181–184. IEEE, 1995.

[33] Carol L Krumhansl. The psychological representation of musical pitch in a tonal context. *Cognitive psychology*, 11(3):346–374, 1979.

[34] Carol L Krumhansl. *Cognitive foundations of musical pitch*. Oxford University Press, 2001.

[35] Carol L Krumhansl. The cognition of tonality–as we know it today. *Journal of New Music Research*, 33(3):253–268, 2004.

[36] Carol L Krumhansl and Edward J Kessler. Tracing the dynamic changes in perceived tonal organization in a spatial representation of musical keys. *Psychological review*, 89(4):334, 1982.

[37] Edith Law and Luis von Ahn. Human computation. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 5(3):1–121, 2011.

[38] Mark Levine. *The jazz theory book*. " O'Reilly Media, Inc.", 2011.

[39] DV Lindley. Regression and correlation analysis. In *Time Series and Statistics*, pages 237–243. Springer, 1990.

[40] José Pedro Magalhaes and W Bas de Haas. Functional modelling of musical harmony: an experience report. In *ACM SIGPLAN Notices*, volume 46, pages 156–162. ACM, 2011.

[41] Ladislav Maršık, Jaroslav Pokornỳ, and Martin Ilcık. Towards a harmonic complexity of musical pieces.

[42] Ladislav Maršík, Jaroslav Pokornỳ, and Martin Ilčík. Improving music classification using harmonic complexity. 2014.

[43] Matthias Mauch, Simon Dixon, Christopher Harte, et al. Discovering chord idioms through beatles and real book songs. 2007.

[44] Matthias Mauch and Mark Levy. Structural change on multiple time scales as a correlate of musical complexity. In *ISMIR*, pages 489–494, 2011.

[45] Tom M Mitchell. Machine learning. 1997. *Burr Ridge, IL: McGraw Hill*, 45(37):870–877, 1997.

[46] Alistair Moffat. Implementing the ppm data compression scheme. *IEEE Transactions on communications*, 38(11):1917–1921, 1990.

[47] Alistair Moffat, Radford M Neal, and Ian H Witten. Arithmetic coding revisited. *ACM Transactions on Information Systems (TOIS)*, 16(3):256–294, 1998.

[48] Daniel Müllensiefen, Bruno Gingras, Jason Musil, and Lauren Stewart. The musicality of non-musicians: an index for assessing musical sophistication in the general population. *PloS one*, 9(2):e89642, 2014.

[49] François Pachet. Surprising harmonies. *International Journal of Computing Anticipatory Systems*, 4:139–161, 1999.

[50] Jean-François Paiement, Douglas Eck, and Samy Bengio. A probabilistic model for chord progressions. In *Proceedings of the Sixth International Conference on Music Information Retrieval (ISMIR)*, number EPFL-CONF-83178, 2005.

[51] Marcus Pearce and Geraint Wiggins. An empirical comparison of the performance of ppm variants on a prediction task with monophonic music. In *Artificial Intelligence and Creativity in Arts and Science Symposium*, 2003.

[52] Jeremy Pickens and Tim Crawford. Harmonic models for polyphonic music retrieval. In *Proceedings of the eleventh international conference on Information and knowledge management*, pages 430–437. ACM, 2002.

[53] Walter Piston. *Harmony*. Norton, 1948.

[54] Lawrence R Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[55] Martin Rohrmeier and Thore Graepel. Comparing feature-based models of harmony. In *Proceedings of the 9th International Symposium on Computer Music Modelling and Retrieval*, pages 357–370. Springer London, 2012.

[56] Peter J Rousseeuw and Annick M Leroy. *Robust regression and outlier detection*, volume 589. John wiley & sons, 2005.

[57] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[58] Ricardo Scholz, Emmanuel Vincent, and Frédéric Bimbot. Robust modeling of musical chord sequences using probabilistic n-grams. In *Acoustics, Speech and Signal Processing, 2009. ICASSP 2009. IEEE International Conference on*, pages 53–56. IEEE, 2009.

[59] Audrey Mae Skaife. The role of complexity and deviation in changing musical taste. 1967.

[60] Sebastian Streich et al. *Music complexity: a multi-faceted description of audio content*. Universitat Pompeu Fabra, 2006.

[61] Erdem Unal, Panayiotis G Georgiou, Shrikanth S Narayanan, and Elaine Chew. Statistical modeling and retrieval of polyphonic music. In *Multimedia Signal Processing, 2007. MMSP 2007. IEEE 9th Workshop on*, pages 405–409. IEEE, 2007.

[62] Christof Weiss and Meinard Müller. Quantifying and visualizing tonal complexity. In *Proceedings of the 9th Conference on Interdisciplinary Musicology (CIM 2014). Berlin, Deutschland*, 2014.

[63] Raymond P Whorley, Geraint A Wiggins, Christophe Rhodes, and Marcus T Pearce. Multiple viewpoint systems: Time complexity and the construction of domains for complex musical viewpoints in the harmonization problem. *Journal of New Music Research*, 42(3):237–266, 2013.

[64] David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.