Skolkovo Institute of Science and Technology

Master's Educational Program 010900 — Applied mathematics and physics

Short-term forecasting of musical compositions using chord sequences

Author: Mikhail Matrosov
Certified by: Vadim Strijov, D. Sc.
Principal Investigator at the CCAS
Accepted by: Mats Hanson, Dean of Education, Skoltech

Moscow

May 2015

Short-term forecasting of musical compositions using chord sequences

by

Mikhail Matrosov

Submitted to the Skolkovo Institute of Science and Technology on May 24, 2015 in Partial Fulfillment of the Requirements for the Degree of Master of Science

ABSTRACT

This work is devoted to predicting musical sequences. Its main focus is constructing a simple yet effective algorithm for predicting music. The developed algorithm uses approach similar to a multiple viewpoint system. The objective is to enhance prediction of a sequence of chords that is treated as multivariate time series of discrete values. A chord is represented as an array of half-tone sounds within one octave. The algorithm utilizes a classifier based on probability distributions over chord sequences that are estimated both on a big training set and some revealed part of the forecasted melody. It shows robust forecasting on Midi50k dataset. The novelty is model selection algorithm and invariant representation of chords. Results of the research can be used as a real time assistant for people who compose music, to generate music on-the-fly for video games and interactive environments like smart homes.

Research (Thesis) Advisor: Vadim Strijov, D. Sc.
Title: Principal Investigator at the CCAS

Contents

Chapter 1

Introduction to the problem of predicting music

Music is an ancient form of art. It is presented in every known culture of humans, and in every culture it is different, with its own peculiarities. But what is common among all of these infinite variety if music is that it connects people, music inspires people in a very special way, that is out of the reach of any other form of art. The word "music" itself is originated from Greek "art of Muses", goddesses of the inspiration of literature, science and arts. Music influences our mood, it can be used to establish comfortable or, on the contrary, a proactive environment for people. It excites some particular zones in a human brain. Researchers know this property and exploit it to study human brain and behavior, connection between emotions and perception, for functional diagnostics and so on.

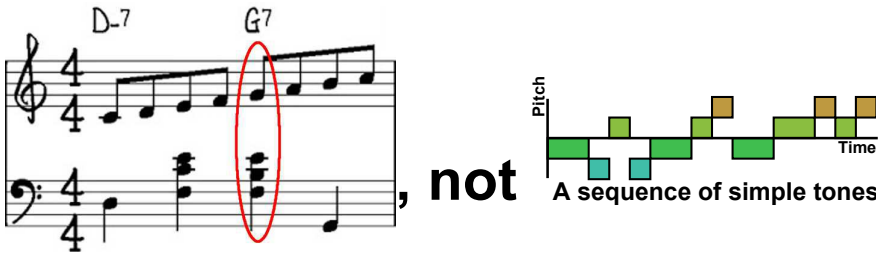

Figure 1.1: A chord sequence (on the left) and a pitch sequence (on the right).

This work is devoted to a problem of predicting music sequence using special chords, like it is illustrated on figure 1.1. As a dataset we use midi files, that can be represented as a stack of simultaneously playing tracks. It can be decomposed into so-called piano roll — a table of volumes for each pitch at each time interval, that further can be converted into special chord representation, described in Chapter 2.1. For training and testing purposes we

use a dataset that consists of 50 000 midi files, randomly grabbed from the Internet. Average length of an obtained sequence is 600 chords, that gives overall about 30 million chords.

Actuality

Nowadays musicians tend to use sequencers instead of traditional music instruments to compose 1.2. These sequencers can be easily paired with a computer, providing additional functionality, like recording notation of played sequence, introducing user interface to more flexible control of a sound, and so on. Since many musicians improvise during their performances, it can be valuable for them to have an assistance, making suggestions about how to play further. Same is about interactive sequencers, that can explain music theory to a beginner and teach how to play his or her composition the best way. As long as this new kind of music composing techniques become more common, such a system, that can suggest a continuation for a melody (i.e. predict it), can become a usual tool for a performing composer. The same method can be applied to produce ambient sound for interactive environments, like games [1], or smart homes [2].

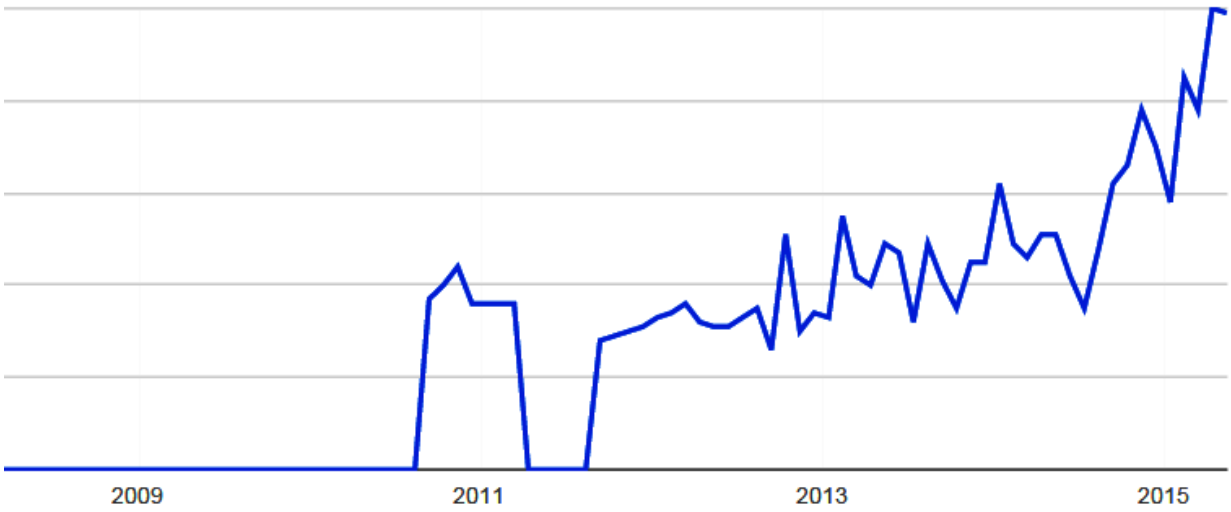

Figure 1.2: Google Trends shows an increasing interest for DJ Pads [3].

Goal

The goal of the research is to build an algorithm that can predict sequence of musical chords of a special kind, one chord at a time (the rest of the sequence can be predicted, using

some sort of propagation: using just predicted chord as an additional element to the input), using some considerations derived from music theory. The algorithm is minimizing some criteria, called error function — difference between a prediction and a ground truth (more detailed description of the error function is in Chapter 2.1). The secondary goal is to collect a comprehensive dataset of midi files, that can be used for testing the proposed algorithm as well as music predicting algorithms in general, and therefore to set a benchmark.

Methods

The problem of prediction time-discrete sequence is a classical machine learning problem. Usually [4] it is reduced to a the problem of minimization of some error function that depends on model parameters, sometimes prediction function can contain several different prediction models, each of that can require different number of parameters depending on model complexity. In this work we apply just one model — composition of multiclass classifiers, but as long as number of these classifiers can vary, number of parameters of that model also can be adjusted. But what's more important is maximum length of used N-grams. N-gram of length up to 16 are used and we claim that $N = 8$ is sufficient in most cases.

In the work we use Hamming distance between sequences of chords to compute error function. The error function is then minimized to obtain optimal parameters. Because amount of data is pretty big, batch processing can be impossible to run, so we use stochastic gradient descend with small batches (typically 50 melodies) on each step.

Novelty

We use more complex representation of a chord based on a special chord decomposition derived from music theory. That gives a significant performance leap, as described in Chapter 5.2. The problem is effectively reduced to classification. We use composition of several naive Bayes classifiers (Section 2.3) for multi-class classification. The point is that we have to use very sparse probability distributions, because amount of possible N-grams is too high to be recorded densely.

Practical value

Prediction of music can be helpful for music writers, suggesting them most common musical tricks and turns, simplifying the process of compositing. This algorithm can also be used to fill in the gaps in a music sequence and noise reduction. It can be used as a foundation for dynamic music composing in real time. For example, in a video game, it can be important to generate music based on a current game state: mood, tension, character disposition. Real-time music composition can bring more comfort into a working or home environment. Suggested algorithm is meant to be applied straight for real cases — it is adapted to be used with usual midi files. The reader can try the algorithm him/herself, since it is publicly available at [5], as well as required data and collected statistics for it [6].

Defendant statements

- Investigated representation of chord sequence as sequence of elements of specially constructed group.
- Developed algorithm of predicting next element in a music sequence.
- The algorithm is tested on Midi50k dataset and compared to other alternative algorithms ([7], [8]) in terms of prediction quality.

Validation

Credibility of the presented results is confirmed by mathematical notion, automated unit-testing, practical test of derived algorithms on a real data, by publications on scientific conferences, including international. Results of the proposed work were presented, discussed and received on the following conferences and seminars:

- M. Matrosov, V. Strijov, A. Matrosov. Short-term forecasting of musical compositions using chord sequences. Conference of International Federation of Operational Research Societies, — July 2014, Barcelona, Spain.
- M. Matrosov, V. Strijov. Short-term forecasting of musical compositions using chord sequences. 57-th Scientific Conference of MIPT. — November 2014, Dolgoprudny, Russia.

- Seminar “Music and Science”, Moscow State Conservatory named for P. I. Tchaikovsky. — January 20, 2015, Moscow, Russia.

Chapter 2

Problem statement for music forecasting

2.1 Related work

In work [7] author applies artificial neural networks to predict pitches and durations of tone in a music composition. Mozer also makes an attempt to predict sets of simple tones (also known as chords). A melody is presented to the network as a vector of ones and zeros, each corresponding to a single tone. The input is presented to the input layer of the network, and the output layer yields the prediction. Both input and output layers of the network contain three parameters of a note: pitch, duration and its harmonic chord accompaniment. Two inner layers — Next Note Distributed (NND) and Next Note Local (NNL) — hold psychophysical and explicit representations, and layer, called Context, holds music history. Overall structure is shown on figure 2.1. The author also experiments with slightly different architectures of neural network.

Mozer uses several specially prepared datasets, each is for a different purpose — learning structure, scales, walk sequences, phrase patterns and etc. One dataset contains real musical compositions.

Difference of the proposed method is mainly in structure of predicting function. Mozer in [7] uses an artificial neural network, it contains context layer and layer that transforms distributed to local representation of chords. Instead, we utilize a feature vectors that explicitly contains history (i.e. several chords preceding the chord to be predicted) and explicit transformation of space of chords. In the proposed work we also utilize a much bigger dataset, that brings a possibility of collecting statistics on algorithm performance.

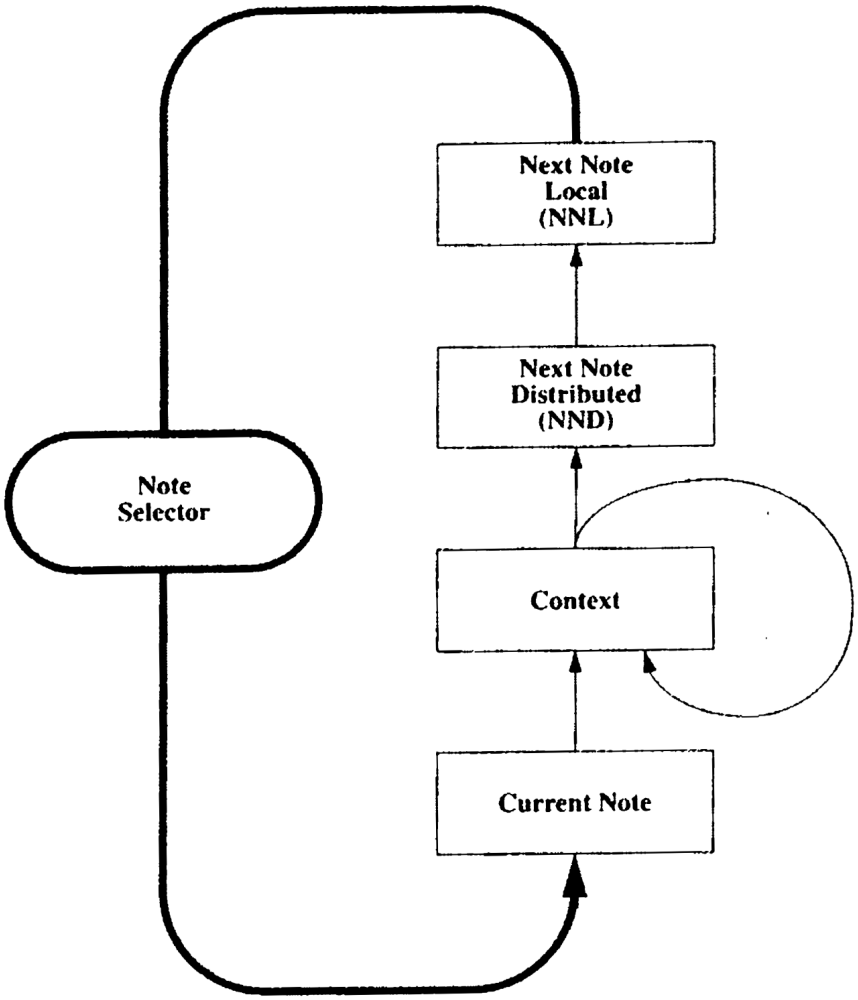

Figure 2.1: Architecture of the neural network in [7]. Rectangles indicate a layer of units, directed lines indicate full connectivity from one layer to another. The selection process is external to the network.

Paper [8] examines prediction of music using a multiple viewpoint system, a collection of independent views of some musical surface. Each view models a specific type of musical phenomena. Machine learning as applied to create models of the general music style and a particular piece of music using both short-term and long-term theories. Then the created model is used to predict the musical sequence. The author measures predictive power of a model by the notion of entropy that is supposed to be an objective quantifiable quality of the model.

Difference of the proposed method is that we hardcoded some musical theory into the algorithm, specifically a chord structure in terms of one octave and relative nature of its pitch — somewhat like engineering approach — comparing to the empirical approach developed

in [8]. On the other hand, that part of the musical theory is the only hardcoded part, the rest (that could also be called syntax) is determined from the collected dataset while learning using statistical analysis.

Book [9] describes basic algorithms that can be used for predicting discrete time series. Different models for a structure of music, as well as explanations and reasoning can be found in [10]. This book is about more or less moderate approaches to the problem of music composing. Book [11] describes boosting techniques. Paper [12] explains different approach to the allied problem — the problem of predicting a discrete time series. Filipenkov and Petrova exploit bunches of discrete time series to find patterns and hidden rules, as well as their evolution over time. Paper [13] illustrates Map/Reduce conceptions on a concrete practical example. In this chapter we explain structure of the space, in which prediction is made, how the prediction is made and a way to measure algorithms performance (error function). As a result the problem is stated in a form of a minimization problem.

2.2 Bayes classifier

For a pair of a vector and a class label (x, y), where $x \in \mathbf{X}$ and $y \in \mathbf{Y}$, the conditional distribution of x, given that its label y, is $P(x|y)$. The difficulties are associated with effective modeling of the probability distributions. In the simplest case, when we have a finite set $\mathbf{X}$, probability can be estimated as

$$P(y|x) = \frac{N_{y|x}}{N_x}. \tag{2.1}$$

The same approach is used to model probability distributions on a set of k-grams of chords:

$$P(x_{i+1}|\{x_{i-k+1}, \ldots, x_i\}) = \frac{N_{\{x_{i-k+1},\ldots,x_i,x_{i+1}\}}}{N_{\{x_{i-k+1},\ldots,x_i\}}}. \tag{2.2}$$

2.3 Composition of classifiers

In the proposed work we use composition of several Bayes classifiers — for a model complexity K, number of classifiers is $2 \times K$, half of them react on 1- to k-grams from the training set, the second half — on the 1- to k-grams from the previous part of the melody.

General definition of composition of classifiers is the following:

$$\mathbb{X} = (x_i, y_i)_{i=1}^{l} \subset X \times Y \text{— training set}, y_i = y * (x_i);$$

$a(x) = C(b(x))$ is an algorithm where

$b : X \to R$ — basic algorithm, $C : R \to Y$ — decision rule, R — space of evaluations.

Definition.

Composition of classifiers.

Composition of basic algorithms $b_1, \ldots, b_T$ is a function

$$a(x) = C\left(F\left(b_1(x), \ldots, b_T(x)\right)\right),$$

where $F : R^T \to R$ is some correcting function.

In our case we use a linear function F, that gives a composition known as weighted voting:

$$F(r_1, \ldots, r_T) = \sum_{i=1}^{T} w_i r_i. \tag{2.3}$$

2.4 Chords structure

Assume we have a group of chords **C**. As soon as representation of the input data is crucial for this problem, let me describe this group in details. Octave consists of 12 semitones. Playing a composition, one can shift all notes into just one octave, and melody will sound

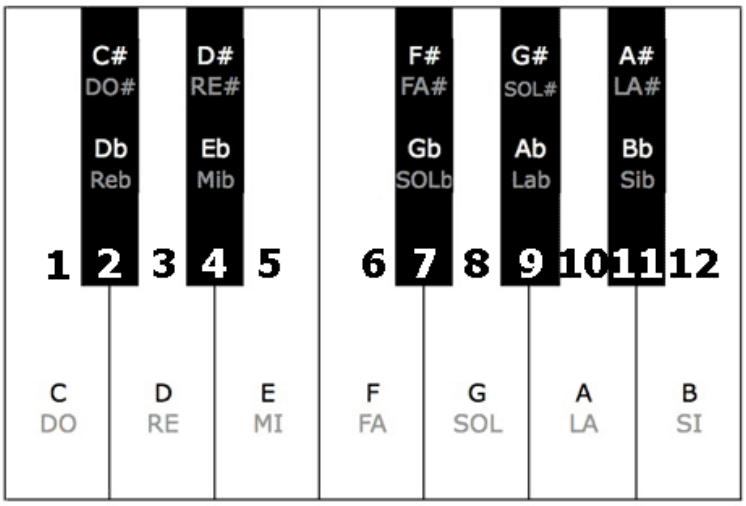

Figure 2.2: Piano keys corresponding to one octave.

almost the same (at least it will remain recognizable by an expert). Each **chord** consists of 1 to 12 simultaneously sounding tones. Chord can be shifted several tones up or down.

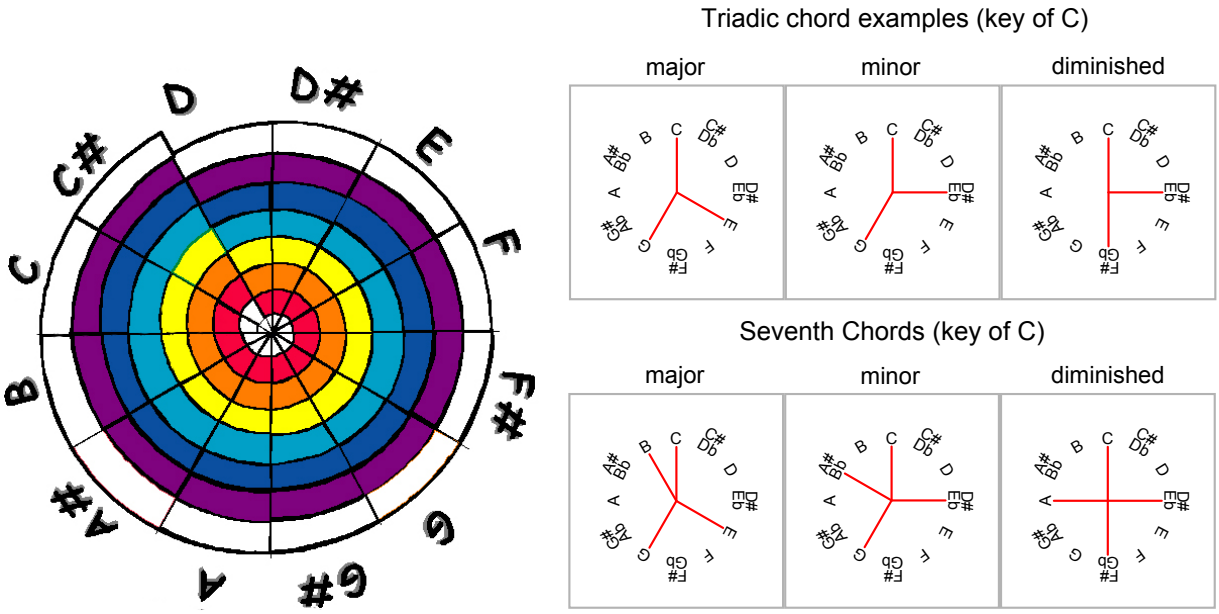

Figure 2.3: Circle of transpositions and six basic chords.

It can also be drawn on a circle, where rotation is changing pitch up or down. Each pitch constellation can be played in 12 different keys (transpositions), except for several symmetric cases (that are pretty rare in real music). Overall it gives us $2^{12} - 1 = 4095$ possible chords. Then a melody can be represented as a sequence of chords.

$$\underset{\text{melody}}{\mathbf{c}} = \underset{\text{sequence}}{\{c_i\}}\ ,\ c_i \in \mathbf{C},$$

$\mathbf{C} = \{1, 2, 3, \ldots, 4095\}$ — space of **chords**. Each chord has its base form (strum $s \in \mathbf{S}$)

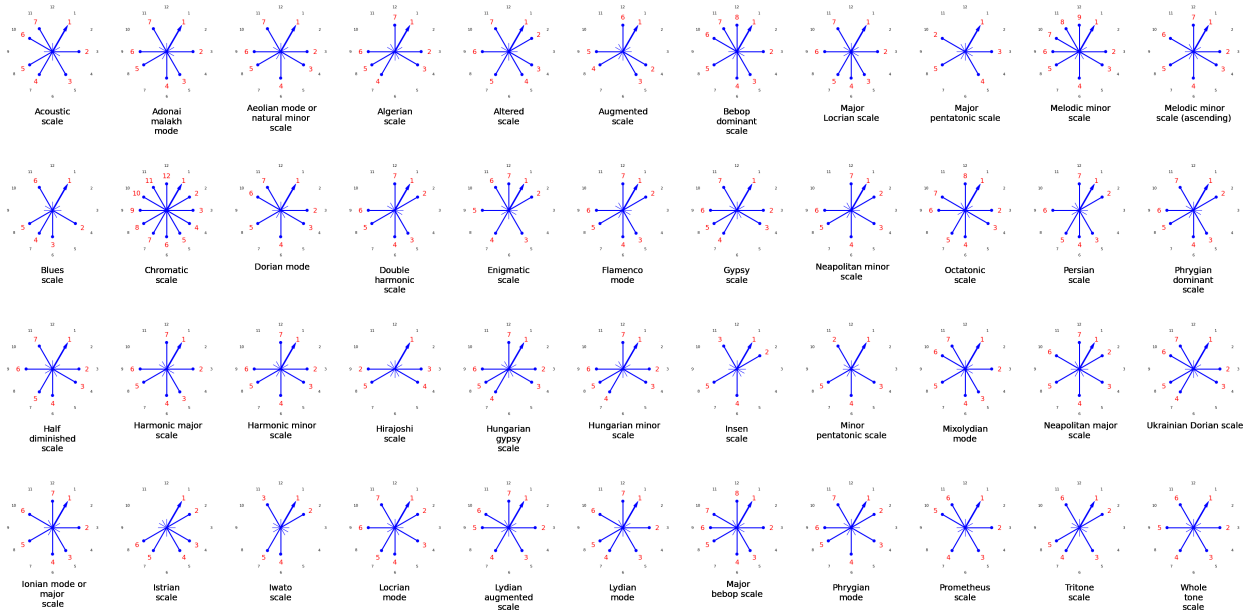

Figure 2.4: Some of the most popular pitch constellations.

and key ($z \in \mathbf{Z}_{12}$). So the whole melody can be represented as a sequence of pairs (s, z):

$$\underset{\text{chord}}{c} = \underset{\text{strum}}{s} \times \underset{\text{key}}{z},$$

$$\underset{\text{melody}}{\mathbf{c}} = \{(s, z)_i\}, \underset{\text{pairs}}{}$$

Multiplication stands for transposition.

Melody can be transposed, so each key is relative to previous keys. That is why we use adjacent differences of keys:

$$r_i = z_i - z_{i-1} \mod 12, \; r_1 = z_1.$$

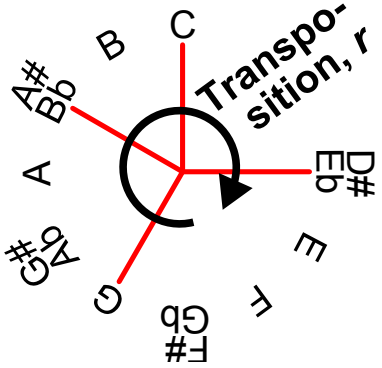

Figure 2.5: Transposition of a chord.

Pair (s, r) is called **element** and denoted as $x \in \mathbf{E}$.

$$\mathbf{C} = \mathbf{S} \times \mathbf{Z}_{12} = \mathbf{S} \times \mathbf{R}_{12} = \mathbf{E},$$

$\mathbf{C}$ - group of chords ($N = 2^{12}$),

$\mathbf{S}$ - group of unique strums ($N = 351$),

$\mathbf{Z}_{12}$ - group of residue classes modulo 12,

$\mathbf{R}_{12}$ - group of differences of Z_{12},

$\mathbf{E}$ - group of elements, that we predict.

There is a bijection between groups $\mathbf{C}$ and $\mathbf{E}$, so every melody can be represented as a sequence of elements

$$\mathbf{x} = \{x_i \in \mathbf{E}, i = 1, 2, 3, \ldots\}.$$

N-gram is a contiguous sequence of N elements $x \in \mathbf{E}$ from a given sequence $\mathbf{x} = \{x_i\}$. N-gram of size 1 (unigram) is just one element $x \in \mathbf{E}$. For example:

$$\mathbf{x}_i^N = \{x_i, x_{i+1}, \ldots, x_{i+N-1}\}.$$

In this work N-grams are used as features describing current point in the music sequence.

$\mathbb{X}$ is a set of melodies: $\mathbb{X} = \{\mathbf{x}_j\}$.

2.5 Prediction function

Prediction is made using weighted sum of several classifiers:

$$x_{i+1} = f(\mathbb{X}, \{x_1, \ldots, x_i\}, \underset{\substack{=\{u_k, v_k\}, \\ k=1,\ldots K}}{\mathbf{w}}) = \arg\max_{e \in \mathbf{E}} \sum_{k=1}^{K} (A_{ek} u_k + B_{ek} v_k), \tag{2.4}$$

$$A_{ek} \propto N \left(\underbrace{\{x_{i-k+2}, \ldots, x_i, e\}}_{k\text{-gram}} \text{ in } \underbrace{\mathbb{X}}_{\text{Training set}} \right),$$

$$B_{ek} \propto N \left(\underbrace{\{x_{i-k+2}, \ldots, x_i, e\}}_{k\text{-gram}} \text{ in } \underbrace{\{x_1, \ldots, x_i\}}_{\text{Part of melody before i+1}} \right),$$

where "$\propto$" means that A_{*k} and B_{*k} are L1-normalized, $x_i \in \mathbf{E}$,

$N(g \text{ in } dataset)$ — number of k-grams $g \in \mathbf{E}^k$ in $dataset$,

$\mathbf{w} = \{u_k, v_k | k = 1, \ldots K\} \in \mathbb{R}^{2K}$ — vector of model parameters (weights), K is model complexity (maximum length of N-grams).

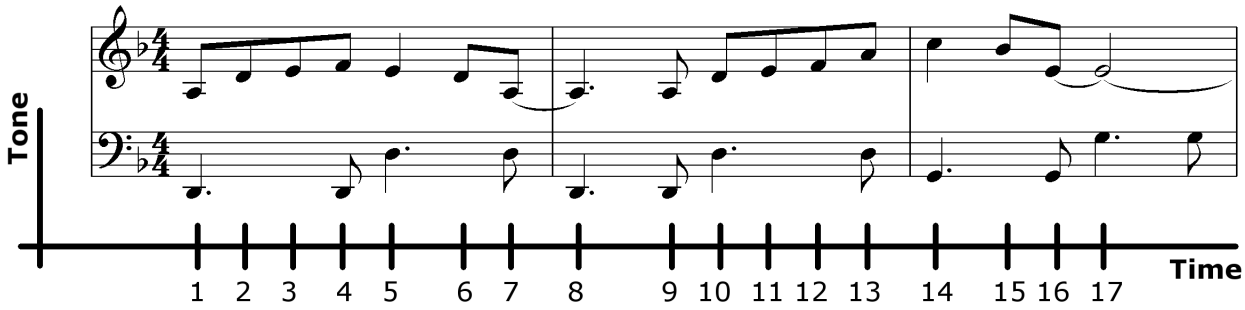

Figure 2.6: Representation of a chord sequence.

2.6 Minimization problem

Function f is a classifier that predicts the next element. Then error function is (i denotes time interval)

$$S(\mathbf{w}, \mathbb{X}) = \sum_{\mathbf{x} \in \mathbb{X}} \sum_{i=1}^{N_{\mathbf{x}}-1} \left[x_{i+1} \neq f(\mathbb{X}, \underbrace{\{x_1, \ldots, x_i\}}_{\text{Prev. part of the melody}}, \mathbf{w}) \right]. \tag{2.5}$$

Brackets stand for 1 if the statement inside is true and 0 if false.

$\mathbb{X}$ is a set of melodies, $\mathbf{w} \in \mathbb{R}^{2K}$ is vector of parameters. For a training dataset $\mathbb{X}$ we would like to find vector $\mathbf{w}$ of algorithm parameters (weights), that minimizes error function:

$$\hat{\mathbf{w}} = \arg \min_{\mathbf{w} \in \mathbb{R}^{2K}} S(\mathbf{w}, \mathbb{X}).$$

2.7 Smooth error function

Error function, as it is written in (2.5), is obviously has steps: every time a prediction for some chord changes from an incorrect to a correct one or inside out, the error function makes a

small leap. Even thought there are many thousands of such tiny steps, this circumstance has an influence on convergence of optimization methods since they are targeted on minimizing differentiable functions. To bear this trouble, one can go with a slightly different, but smooth error function:

$$S_{smooth}(\mathbf{w}, \mathbb{X}) = \sum_{\mathbf{x} \in \mathbb{X}} \sum_{i=1}^{N_{\mathbf{x}}-1} B\left(x_{i+1}, \hat{f}(\mathbb{X}, \underbrace{\{x_1, \ldots, x_i\}}_{\text{Prev. part of the melody}}, \mathbf{w}) \right), \tag{2.6}$$

$$B(x, \hat{f}) = \tanh(-s \frac{M_1 - M_2}{|M_1 + M_2|} \cdot e),$$

where M_1 and M_2 are the first and second maximum elements of the probability distribution $\hat{f}$ from f in (2.4), s — is a scale parameter, and e is 1 if the prediction is correct and -1 otherwise.

The rationale for such a function is basically our need for a function, that is substantial to comparing the result of forecasting. For example, let's consider introducing something like a Hellinger distance between a predicted probability distribution P on the next chord and a ground truth probability distribution $G = (0, \ldots, 0, 1, 0, \ldots, 0)^T$. Probability distribution P is going to have a set of almost equal numbers for the most probable next chords, and only because we minimizing the error function, we are able to get more correct predictions. It also means, that a slightly different vector P can give us a very different prediction result — because it can have another maximum element. Anyway, these two distributions are much closer to each other then to the ground-truth vector G, that makes this metric inconsistent and not informative for our purpose.

Another example — comparing a maximum element from the distribution to an average or a median from elements of the vector P. Counter-example for this type of distances would be

$$P_1 = (0.5, 0.49, 0.001, \ldots, 0.001) \quad \text{and}$$

$$P_2 = (0.2, 0.1, \ldots, 0.1).$$

In the first case there are two salient candidates (1 and 2) for the next chord, which both look quite the same. Distance for the maximum element both from average and median is quite big, which could lead one to a thought, that it's a good prediction. But in reality, this prediction is very unstable, it is really close to the boundary — every little fluctuation in initial parameters will shift maximum to another element and will make the prediction incorrect. On the other hand, distribution P_2 is more regular, meaning there are no salient

maximums. Maximum can be just several percent bigger than average, but obviously it is much harder to shift from one result to another.

This reasoning is leading us to the proposed smoothing function.

Theorem.

Smoothed error function has nearly the same minimum as the original one, i. e.

$$x + \Delta x = \underset{\mathbf{w}}{\operatorname{argmin}}\, S_{smooth}(\mathbf{w}, \mathbb{X}), \tag{2.7}$$

$$x = \underset{\mathbf{w}}{\operatorname{argmin}}\, S(\mathbf{w}, \mathbb{X}), \tag{2.8}$$

$$\forall s > 0 \;\; \exists \varepsilon_s > 0 : \quad \Delta x < \varepsilon_s. \tag{2.9}$$

In case $M_1 = M_2$ the argument of tanh is 0, so sign of e doesn't matter. Now from the definition of error function (2.4) we easily derive:

$$\begin{bmatrix} M_1 \\ M_2 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix} \cdot \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \tag{2.10}$$

From there follows:

$$B(x, \hat{f}) = \tanh(s\frac{k_1 w_1 - k_2 w_2}{k_1 w_1 + k_2 w_2}) \cdot e, \tag{2.11}$$

where k_1 and k_2 are some coefficients combined from the previous equation. Calculating a derivative from this function reveals:

$$B'_{w_1} = \left[1 - \tanh^2\left(s\frac{k_1 w_1 - k_2 w_2}{k_1 w_1 + k_2 w_2}\right)\right] \frac{2sk_1k_2w_2}{(k_1 w_1 + k_2 w_2)^2} \cdot e. \tag{2.12}$$

Effectively, this function transforms into a clipped slope:

$$B'_{w_1}(x, \hat{f}) = \begin{cases} 0, & \text{if far away from from the boundary,} \\ e \cdot s \cdot k(w_1, w_2), & \text{otherwise.} \end{cases}$$

It means, that for $s \to +\infty$, function $S_{smooth}(x)$ is effectively the same as $S(x)$. The smaller value of s, the smoother function becomes, as it is shown on figure 2.7. Oversmoothed

function will give less accurate results, but the optimization process will finish faster, because there will be no plateaus. The theorem's terms can be reformulated as:

$$\sum\sum B'(x, s \to +\infty) = 0, \tag{2.13}$$

$$\sum\sum B'(x + \Delta x, s) = 0, \tag{2.14}$$

$$\forall s > 0 \;\; \exists \varepsilon_s > 0 : \quad \Delta x < \varepsilon_s. \tag{2.15}$$

Summing up all the derivatives for B we determine, that the minimum is not far away from its original position because B is a continuous function bounded in $[-1, 1]$. ■

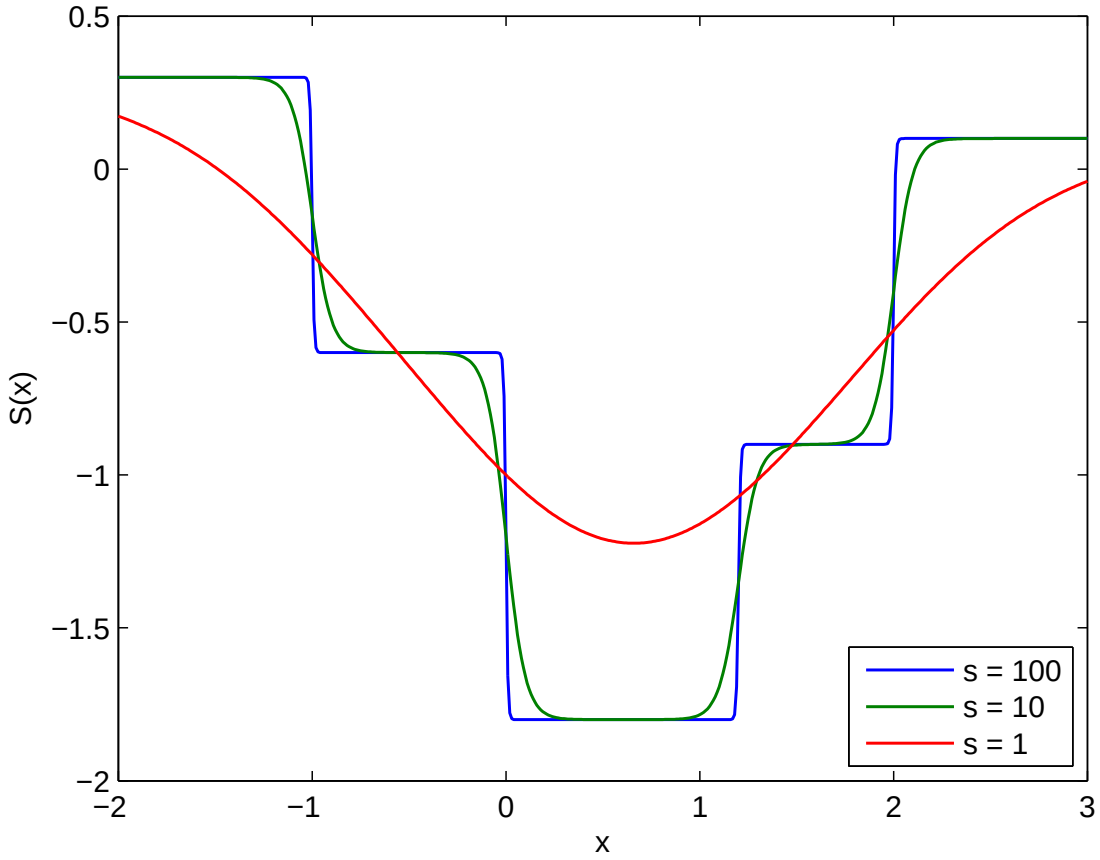

Figure 2.7: Smooth error function for different values of parameter s.

On the figure 2.8 shown projections of error function $S(w)$ from 2.5 and smooth error function 2.6. As you see, in most cases, minimums are very close to each other, and just in one case (for w_9) minimums are distinguishably different, so these trends proof that original

and smoothed functions have closely situated optimums. These plots are build just for 5 songs from the dataset, and a closeup of region of the minimum is given to illustrate the difference between locations of smooth and raw error functions.

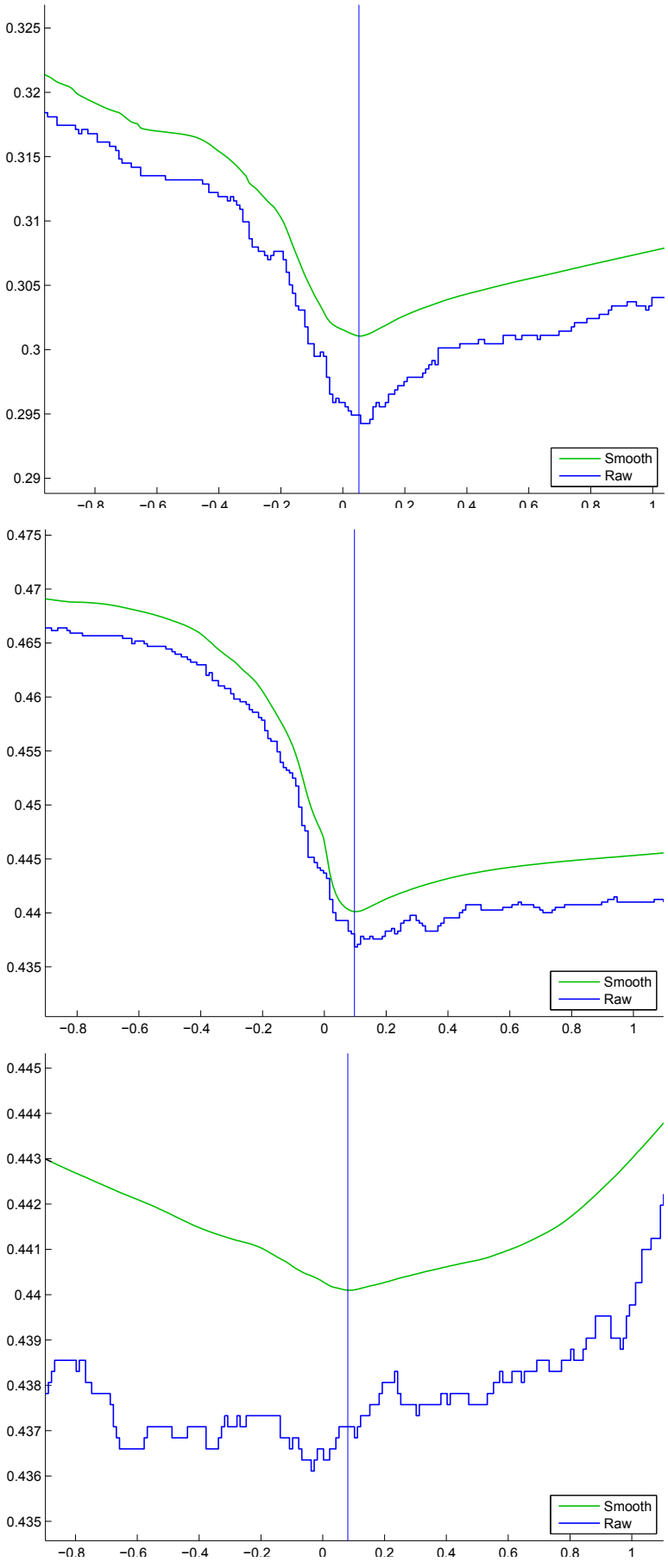

Figure 2.8: Projections of the error function $S(w)$ to planes $S(w_3)$, $S(w_{18})$ and $S(w_1)$.

Chapter 3

Midi50k Dataset

Here is a description of the dataset, that was used to train and test the proposed algorithm. We explain how the dataset was constructed and used, what parameters does it have.

The dataset was collected specially for this work. It consists of 49 949 midi-files, downloaded from the Internet. They are of different authors, of different genres and from different time. The dataset can be found at [6]. Overall project description and scripts to collect this dataset are stored on the Sourceforge [5].

3.1 Midi-files

MIDI (Musical Instrument Digital Interface) "is a technical standard that describes a protocol, digital interface and connectors and allows a wide variety of electronic musical instruments, computers and other related devices to connect and communicate with one another." [14] It carries events, that specify notation, pitch, velocity, volume and other parameters for multiple devices. These messages can be received by other devices which generate sound accordingly. The same stream of events can also be stored to be played back later or edited.

Midi-file is an audio file, that contains not the audio wave itself, but instructions to generate it — recorded stream of events from the previous paragraph. It can be roughly understood as musical notation for a particular song. Thus it is easy to parse and to operate.

Midi-files are very small (about 40 KB for an average file, that can be compressed up to 3 KB) comparing to MP3 (around 5 MB) or lossless (FLAC, around 50 MB). Although they do not contain any information about human voice, it is impossible to record singing with

midi-file. Since we predict only music, not trying to predict lyrics, midi-files are perfect for our purposes.

Midi-files are very common inside audio studies during creation and mastering musical compositions. But it is not a common thing for a composer to share them. Midi-files are also created by enthusiasts or can be automatically generated from an audio stream with some special software. But because midi-files cannot contain any human voice, they are much less spread nowadays, comparing to other audio files, like MP3.

3.2 Collecting method

One can find a whole bunch of midi-files on the Internet, but usually they are distributed in smaller collections. A Python script [15] was written to grab midi-files from the Internet, at the same time storing their title and authors name. The script works as a usual crawler — it takes input links to web-pages, checks them looking for links to midi-files, that can be downloaded. Other links pointing to the same web-site are stored and visited later. We do not visit the same page or download the same file twice. For that purpose we maintain a set of already visited links. Also we have to store a queue of links that we are to visit — that were found but not visited by the script yet.

3.3 Obtaining related data

Midi-files do not contain any information about the author. Since that we have to obtain all the related data (genre, year of creation, place of creation, etc) ourselves. For that purpose another Python script was written. For every author's name it performs an Internet search (using Bing search engine) in order to retrieve information on that author. Then it parses Wikipedia page devoted to the author. All the data is stored in an electronic table, checked and manually edited later. The table contains the following columns:

1. Author's name.
2. Number of songs of that author in the collection.
3. First year, that the author was active.
4. Last year, that the author was active (9000 if active by the time collection is created).

Table 3.1: An excerpt from the mined database, showing dataset structure.

Name	N Songs	Active from	Active until	Genre	Labels	City	State	Country
led zeppelin	186	1985	2007	hard rock, blues rock, folk rock, heavy metal	atlantic, swan song	london		u.k.
bach johann sebastian	166	1700	1750	classical		eisenach	saxe-eisenach	germany
dadi marcel	92	1971	1996	chanson				france
madonna	48	1979	9000	pop, rock, electronic	sire, warner bros., live nation, maverick, interscope	bay city	michigan	u.s.

5. Comma separated list of genres
6. Comma separated list of labels/studios, that author was published under (if applicable).
7. City of origin.
8. State of origin.
9. Country of origin.

Table 3.1 is an excerpt from the mined database.

3.4 Collection details

Overall, there are 49 949 midi-files, of 6918 different authors/compositors/music bands. Each author is labeled with several genres. Figure 3.1 shows distribution of authors by a number of songs in the collection, written by this author. There are 40 basic genres, like “rock”,

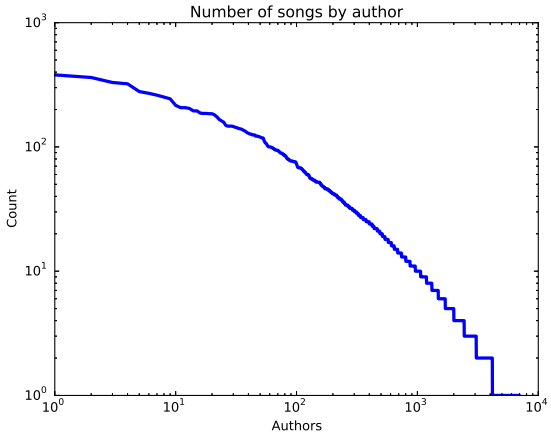

Figure 3.1: Number of songs varies for different authors. This dependence is shown on the trend.

"classical", "blues", etc, and 897 detailed genres, e.g. "alternative rock", "nu metal" and so on. Number of songs of each author is related to its popularity in contemporary culture. On the side is the distribution of number of songs by author, below are distributions by several popular genres, refer to figure 3.2.

Genre is labeled with a string, usually of one or two words, like "'alternative rock", "nu metal" or "horror punk" or "hip hop". On the graphics below, rock means any possible kind of rock. Same is about other genres on the plots.

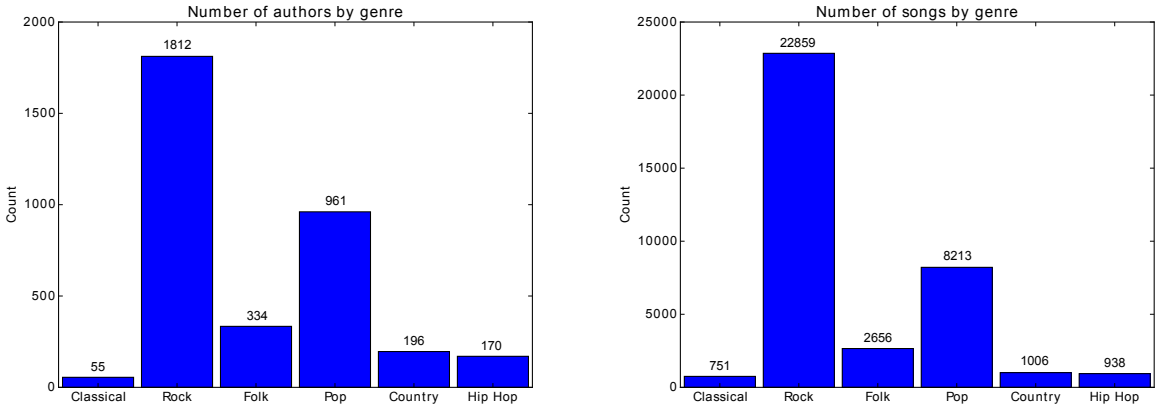

Figure 3.2: Number of authors and songs by genre. Only 6 of out 40 basic genres are shown.

3.5 Naming conversion

Filenames and table entries have different naming conventions. It is due to historical reasons. Author's name can be retrieved from a midi filename with the following Python function:

```
# convert midi filename to obtain author's name
def getAuthorName(subject):
    author = subject.split("-")[0] # part before "-"
    author = " ".join(author.split("_")) # split by "_"
    author = re.sub("(?=[A-Z])", " ", author) # split by letter-case
    author = re.sub(" +", " ", author) # remove double spaces
    return author.strip().lower()
```

It takes part of the input string, that stands before hyphen "-", substitutes symbols "_"with spaces, then splits words by understanding letter case, and finally, all double spaces are removed and the string is stripped of unnecessary characters on its beginning and ending.

3.6 Train/Test dataset division

For evaluation purposes full dataset $\mathbb{X}_0$ (50 000 melodies) was being split several times in two pieces of different size. Each time splitting was performed on a random basis — from the dataset was selected a subset (without returns) of requested size M.

$$\mathbb{X}_{\text{training}} \subset \mathbb{X}_0,$$

$$|\mathbb{X}_{\text{training}}| = M.$$

To test the algorithm we use the rest of the dataset:

$$\mathbb{X}_{\text{testing}} = \mathbb{X}_0 \setminus \mathbb{X}_{\text{training}}.$$

Each midi file was converted to a sequence of chords $\mathbf{c} = \{c_i\},\ c_i \in \mathbf{C}$ with the following steps:

- open midi file as a piano roll,
- remove percussion part,
- quantize time with rate $2/tempo$,

- strip octave number ($new\ pitch = pitch \mod 12$).

Average midi file contains sequence of **600 chords**, that gives **30 million** chords overall.

Melody is a sequence of elements (index is time): $\mathbf{x} = \{x_i\},\ x_i \in \mathbf{E}$.

$\mathbb{X}$ is a set of melodies: $\mathbb{X} = \{\mathbf{x}_j\}$.

Chapter 4

Implementation of the method

This chapter is devoted to the actual method implementation: how the data is organized, algorithms for dealing with this data, used optimizations that result in a more efficient code. Most of the algorithm is written in MATLAB and available at [5].

4.1 Data structures

4.1.1 Midi files

Input is represented as a set of MIDI files with extension ".mid" — generic midi files. There are no special requirements for these files in any sense.

4.1.2 Probability distribution

Probability distribution is a basic structure for the prediction system.

pd	Probability distribution among elements.	
containers.Map	string(n) → double	For each combination of elements, given as a string, contains probability to find it in the training set.

4.1.3 Reshaped probability distribution

Reshaped probability distribution is used to dramatically increase performance during prediction phase.

rpd	Reshaped probability distribution among elements.	
containers.Map	string(n-1) → mat(ELG_SZ, 1)	For each combination of elements, given as a string, contains probability distribution on the next element in the sequence.

4.1.4 Feature vector

Set of probability distributions used for boosting.

fv	Set of probability distributions.	
sparse mat	[ELG_SZ, 2*LDEPTH]	Matrix with probabilities. ELG_SZ — size of elements group (basically, 4212), LDEPTH — model complexity (lookup depth), default is 8.

4.2 Project architecture

Overall system architecture is described with the IDEF0 format on the graphics below. Project consists of the following modules:

- Dataset builder — includes Python scripts to look for midi files in the Internet, fetch metadata, and to transform datafiles to MATLAB-readable files.
- Midi parsing — converting a midi file to a sequence of chords.
- Learning part of the algorithm.
- Predicting model.
- Optimization — minimizes error function (using stochastic gradient descend).
- Statistics and visualization.

IDEF0 diagrams of the project are shown on figures 4.1 and 4.2.

4.3 Parsing midi-files

For parsing midi files in MATLAB was used Ken Schuttle's library [16]. He included some very basic functions, like reading and writing midi files, parsing it as a piano roll and so on.

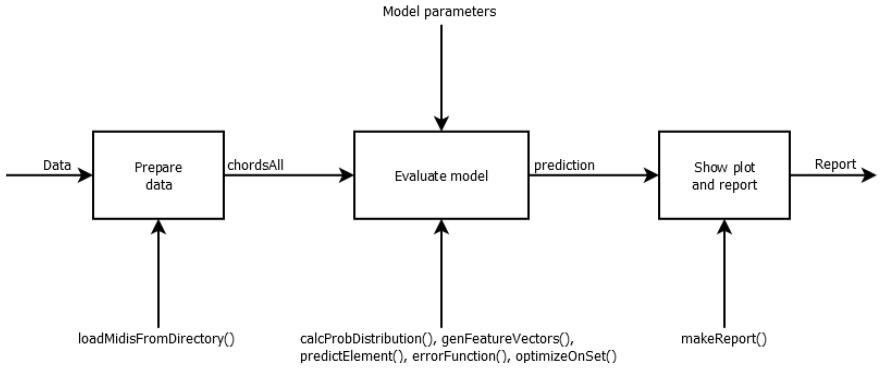

Figure 4.1: A1 level diagram showing system architecture in the IDEF0 format.

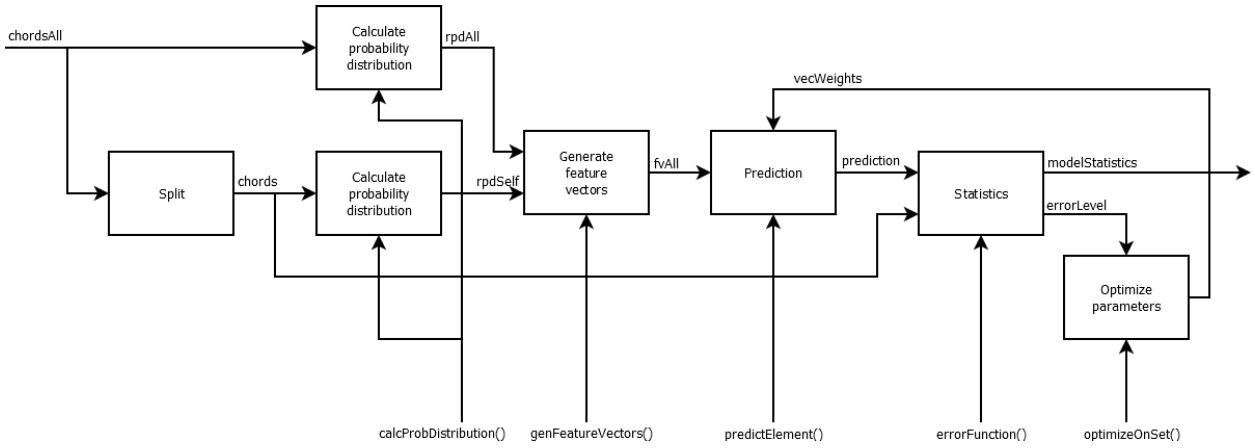

Figure 4.2: A2 level diagram expanding system architecture of model evaluation module in the IDEF0 format.

On top of this library was written set of functions that converts a piano roll to a sequence of chords.

Using a specially generated scheme, group of chords is decomposed into a product of two groups — strums and keys:

```
stMap = genStrumsMap();
elements = chords2elements( chords, stMap );
```

Variable stMap is a matrix of size 2×4095, first row contains corresponding strum number (1 to 351), second row — basic key of the chord (0 to 11). Part of it's contents con be seen in table 4.1.

This table is generated by the following function (in MATLAB notation):

```
function [strumsTonesMap] = genStrumsMap()
    chords = 1:4095; % all possible combinations of 12 bits
```

Table 4.1: Contents of matrix stMap.

1	1	2	1	3	2	4	1	5	...	350	219	320	340	350	320	350	350	351
0	11	0	10	0	11	0	9	0		8	9	9	9	9	10	10	11	0

```
    strums = zeros(1,4095); % maps the set of chords into the set of strums
    tones = zeros(1,4095); % maps into the set of tones
    n = 1;
    while numel(chords)>0
        ch = chords(1);
        if ismember(ch, chords)
            arr = zeros(12,1);
            for j=1:12
                arr(j)=rotateChord(ch,j);
            end
            chords = setdiff(chords, arr);
            strums(arr) = n; % ch;
            n = n+1;
            tones(arr) = 11:-1:0;
        end
    end
    strumsTonesMap = [strums;tones];
end
```

The idea is similar to the sieve of Eratosthenes. First set of all 4095 chords is created, then we select an element from this set, get all 12 rotations of it (meaning the same chord but for all possible keys), and remove it from the set. This process continues, until all chords are represented as a product strum×key.

4.4 Map container

Assuming we have model size K, meaning there are N-grams for N up to K, we need to store feature vectors of huge dimensionality $2^{12\times K} \approx 6.3 \cdot 10^{57}$ (for $K = 16$). It is way bigger even

than 64-bit wide integers, that is why we have to use associative arrays (so-called maps). Luckily, these vectors are extremely sparse (no more than 250k non-zeros).

4.5 Calculating probability distribution

Map containers are hard to deal with — they have really bulky keys stored in a hash map. The point is that updating a map with a single entry is as hard as merging two maps. It also takes a lot of space when not reduced. In the second case we can keep track of the smallest entries, that we can omit early the collecting stage without risk of loosing precision. It makes preferable to split dataset into several parts, collecting statistics in each one separately and then merging results. So, overall we developed a procedure, similar to Map/Reduce strategy [13], but recursive:

- If the dataset is too big, split it into two parts.
- For each part calculate probability distribution.
- Merge probability distributions (Reduce stage).
- Remove small values (Map stage).
- Return decimated probability distribution.

4.6 Minimization

4.6.1 Stochastic gradient descend

One evaluation of predicting and error function for the whole Midi50k dataset can take 100 hours of machine time (dual-core Intel P6100 @ 2.0 GHz). Therefore it is better to make small steps for just a small part (bunch) of the training set [17]. One bunch is typically 100 melodies (random subset) comparing to 10 000 usual dataset and it can be evaluated much faster because it fits into RAM.

Each step is implemented using Levenberg-Marquardt algorithm, also known as the damped least-squares (DLS) method. Once we found optimal parameters for this small bunch, the current position is updated with a decreasing weight ($\alpha = \frac{1}{\text{no. of iteration}}$). We have pretty big bunches of data, that's why convergence trend is relatively smooth:

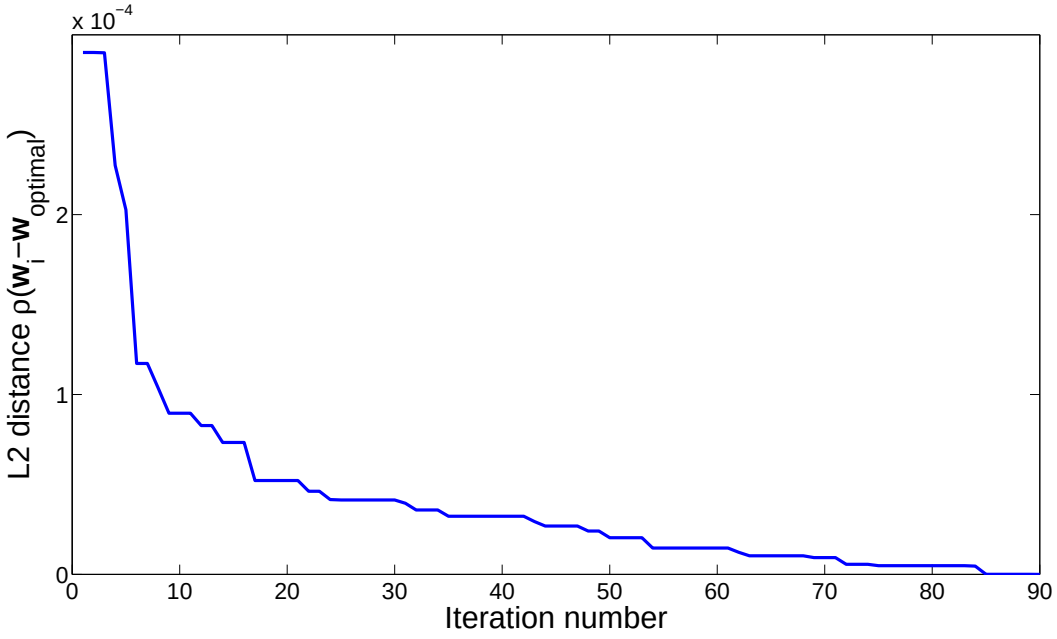

Figure 4.3: Convergence of stochastic gradient descend.

Stochastic gradient descend is commonly used in cases when amount of data is overwhelmingly big — for example, training artificial neural networks. This method has proven its effectiveness and converges almost surely, and if the objective function is convex or pseudoconvex — almost surely to a global minimum.

4.6.2 Levenberg-Marquardt algorithm (LMA)

It is a well-known algorithm for solving non-linear least-squares problems, like curve fitting [18]. Basically, LMA is an interpolation between a usual gradient descend and Gauss-Newton minimization. It is more stable and robust in most cases and tends to find minimum even if the starting point is set far away from the minimum. However, it finds only a local minimum, which is also common for most minimization algorithms.

If the problem is stated as

$$S(\beta) = \sum_{i=1}^{m} [y_i - f(x_i, \beta)]^2 \to \min . \tag{4.1}$$

LMA is an iterative algorithm. It starting point user gives an initial position — vector β. It can be random, specified by user or derived from some intuitive estimations. Then on each iteration we update vector β, adding vector δ, obtained from the following equation:

$$\left(J^T J + \lambda \mathrm{diag}(J^T J)\right) \delta = J^T \left[y - f(\beta)\right], \tag{4.2}$$

Table 4.2: Computational complexity and evaluation time for different parts of the project.

Stage	Reading midi files	Calculating probability distribution	Reducing probability distribution	Reshaping probability distribution	Generating feature vectors	Weights optimization	Quality test
Complexity	$O(n)$	$O(nlogn)$	$O(k)$	$O(k)$	$O(n)$	$O(n)$	$O(n)$
Time	20 min	15 min	$<$ 1 min	$<$ 1 min	30 min	$\sim$1 hour	$\sim$1 hour

Where $J = \frac{\partial f(x_i,\beta)}{\partial \beta}$ is a Jacobian of the minimized function f and λ is some damping factor.

4.6.3 Stability enhancements

Because we have a function, that has finite steps and multiple local minimums, and because the result of minimization depends on the initial point, we would like to even further improve stability of the optimization algorithm. For that sake we introduce damping of the vector of parameters:

$$\begin{aligned} \beta_i &= \text{argmin}(f, \hat{\beta}_{i-1}), \\ \hat{\beta}_i &= \beta_i(\alpha - 1) + \hat{\beta}_{i-1}\alpha, \end{aligned}$$

where α is another damping factor, that is also exponentially decreased from iteration to iteration.

4.7 Computational complexity

System consists of 7 separate parts — stages. Performance and complexity of each stage is shown in the table 4.2 (for a smaller dataset, just 350 midi files, only to show order of magnitudes).

Overall complexity is $O_1(n \log n) + O_2(n)$ where $O_1 \ll O_2$. One midi-file can be predicted in about 3 seconds, and the most time-consuming parts of this process are reading the actual midi file (due to inaccurate design of used MATLAB lib to read midi files) and generating feature vectors, because it requires lots of queries to the probability distribution containers.Map. Memory consuming is within reasonable for all the stages, except for generating feature vectors — it easily finishes all the memory (that is limited to 2 GB for 32-bit JVM). Here is a trade-off between memory consumption and time required for optimizing the

weights:you can fit into only about 100 MB of memory (generating probability distributions on-the-fly) but in this case optimization will take about 30 hours on a set of 350 midi files, and several months for the full Midi50k dataset.

Chapter 5

Computational experiment

The chapter describes results obtained by applying the proposed algorithm to the Midi50k dataset, gives some understanding of the properties of the method and about optimal parameters that can be fed to the system.

5.1 Visualization of the results

Let's start with an interesting fact. Probability distributions of N-grams with different N. Number of events is a number of occurrences found in a set of 50 000 midi files. Slope coefficient is about -0.6, distribution is similar to distribution of words in a natural language (Zipf's law [19]). This is a pretty interesting phenomenon, it explains similarities between music natural language and also makes a point why a technique based on N-grams can work for the purpose of predicting music. The trend for 1-grams is decaying faster for bigger ordinal numbers of chords, because there are only 4095 of chords, an exact number, unlike in a situation with words from a natural language. And it is clearly visible, how increasing number of possible words (combinations, N-grams in this case) leads to a more linear shape of the trend (in a double logarithmic scale, of-course).

Picture 5.2 shows a heat map of distribution probability of elements $\mathbf{E}$. Number of events is a number of occurrences found in a set of 50 000 midi files. Order of strums (horizontal axis) is arbitrary — result of representing a chord as a pair (s, r). Notice, that color scale is logarithmic, meaning that some chords are more common than other by a factor of 10^6.

Forecasting quality. Number of parameters is 16, training set is 50 000 melodies. Average prediction error is **0.42** (meaning 58% of successfully predicted elements $x \in \mathbf{E}$). There are also melodies that were forecasted on 100%, as well as melodies forecasted poorly (<5%)

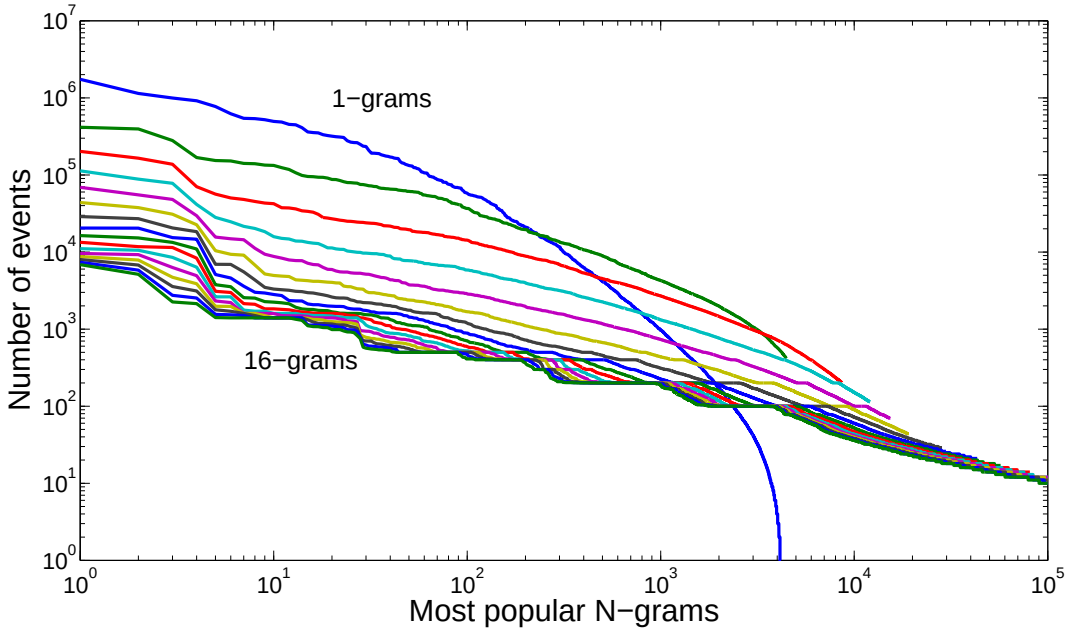

Figure 5.1: Probability distribution on N-grams showing Zipf's law.

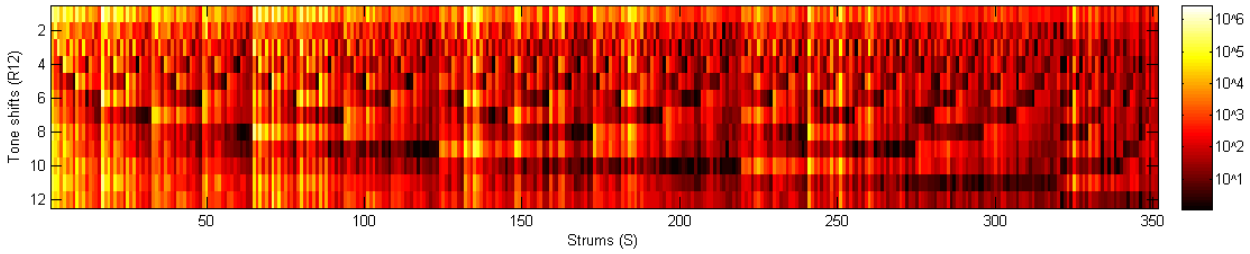

Figure 5.2: Heat map of elements.

Distribution looks like a usual binomial distribution — it doesn't have significant fluctuations and has only one maximum, which approximately is 0.4, as shown on Figure 5.3.

There are several songs with corresponding prediction quality in the table. Pop-songs usually are better predicted then the rest of the collection. Table 5.1 shows just several compositions, sorted by the performance of the algorithm, applied to them.

There is also a dependence from amount of data. As you on the figure 5.4, prediction quality on a training set and on a test set are converging to a common value. It is also noticeable, that 1000 midi files is somewhat like a minimum size of the dataset, that allows relatively good training of the algorithm.

Graph 5.5 shows error function vs number of parameters K ($\mathbf{w} \in \mathbb{R}^{2K}$), on a test set. Note a significant leap after $K = 5$ and almost no difference for K bigger than 8. It is because 5 is basically a full musical measure, plus one additional chord, that is required to

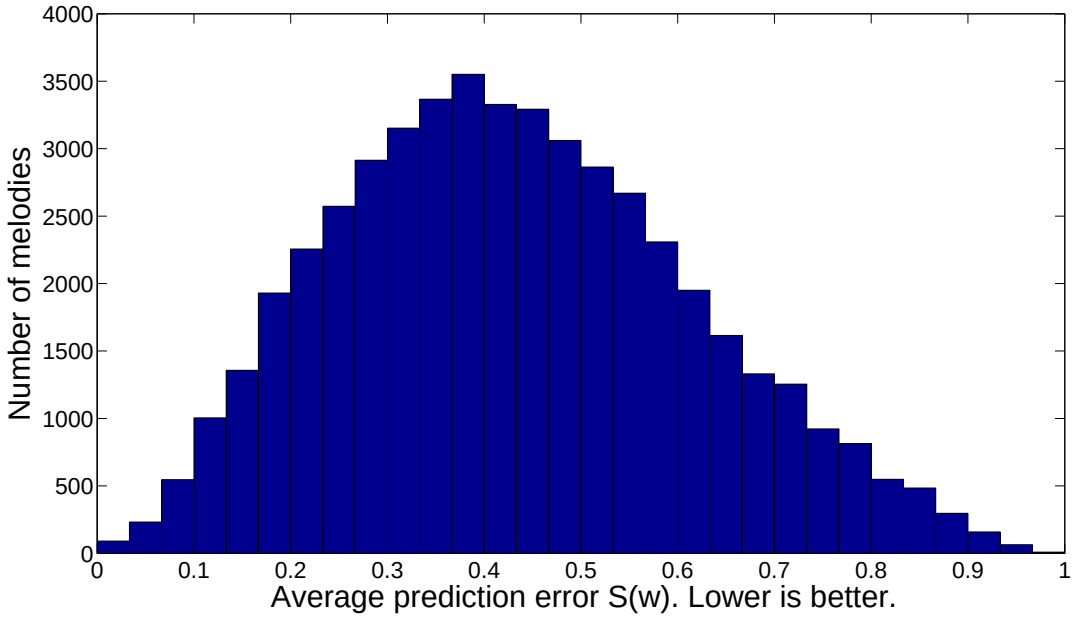

Figure 5.3: Some songs are easy to predict, some other songs are very unpredictable.

understand the context. But additional 4 chords are enough to identify most of the contexts, that's why error isn't really reduced by even further increasing of the K parameter.

Figure 5.6 is demonstrating a visualized output of the algorithm. It is a fragment from Beethoven's Silence. The chart is stylized in such a way that the reader can better understand the input and the output of the algorithm. Input is represented as empty circles, dots represent the prediction, vertical axis is a pitch and horizontal axis is time.

It can appear that different genres will tend to have different sets of chords or 2-grams of chords, but in reality, it doesn't look like that. On the contrary, all the genres seem to use similar sequences of chords. On the figure 5.7 melodies are shown as data points on a 2D plane, which is the projection from space of frequency distribution over 2-grams. The projection consists of two first principal components, obtained using PCA method [20]. Blues, classical and hip-hop music was chosen because they are the most distinguishable from each other, and their frequencies in the Midi50k dataset are comparable (538, 324 and 45 songs respectively). Genres doesn't form clusters in this space, as it is shown on the figure. A rationalization for this is that music genre is defined mostly by its rhythm and spectrum of lower frequencies of the melody (bass part), thats why the proposed model is insensitive to differences between genres.

Table 5.1: Compositions sorted by prediction quality, showed by the proposed algorithm.

Katy Perry California Gurls	90.5%
The Twelve Days of Christmas	88.5%
The First Noel	88.5%
...	...
Tchaikovsky, op. 37	25.3%
Bodysoul	21.9%
Godrest	18.2%
AVERAGE	58.0%

Table 5.2: Performance of the proposed algorithm.

Quality	Mozer[7]	Conklin[7]	Proposed
Main idea	Neural network	Music patters	Bayes classifiers
Chords	—	40%	**58.0%**
Pitches	93%	**95%**	92.5%
Durations	**90%**	75%	—
Datasize	20	4500	50 000

5.2 Comparison with existing methods

This set of experiments was conducted in order to compare performance of algorithms. As you can see in the table below, the proposed algorithm scores its ancestors in terms of predicting chords as a whole. The results are also more reliable, since we used more data to test our method. Table 5.2 shows a comparison between the proposed algorithm and its ancestors.

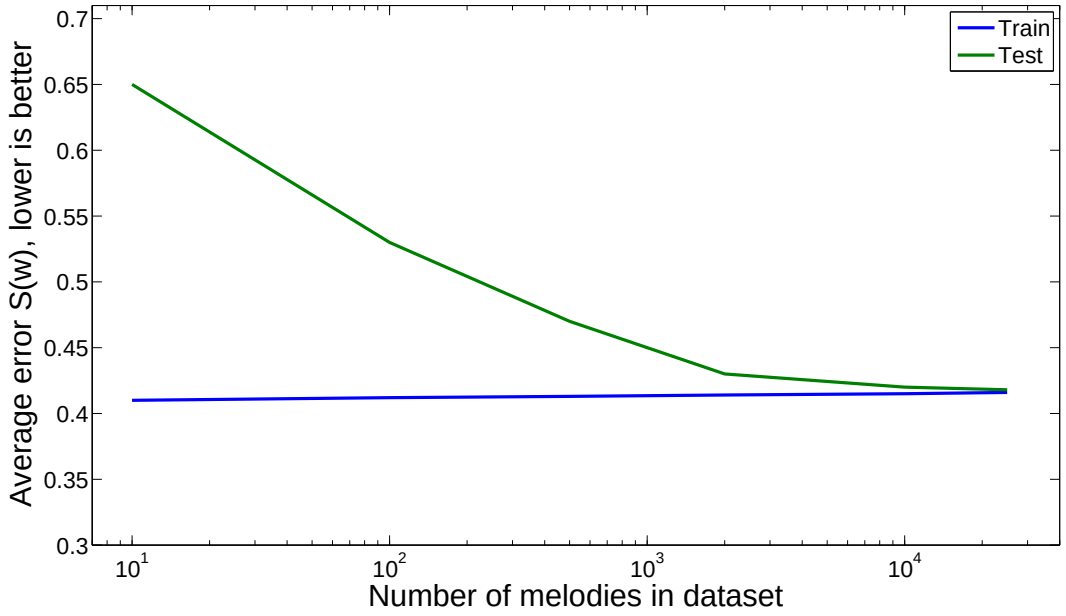

Figure 5.4: Prediction quality on testing and training sets depending on the size of the set.

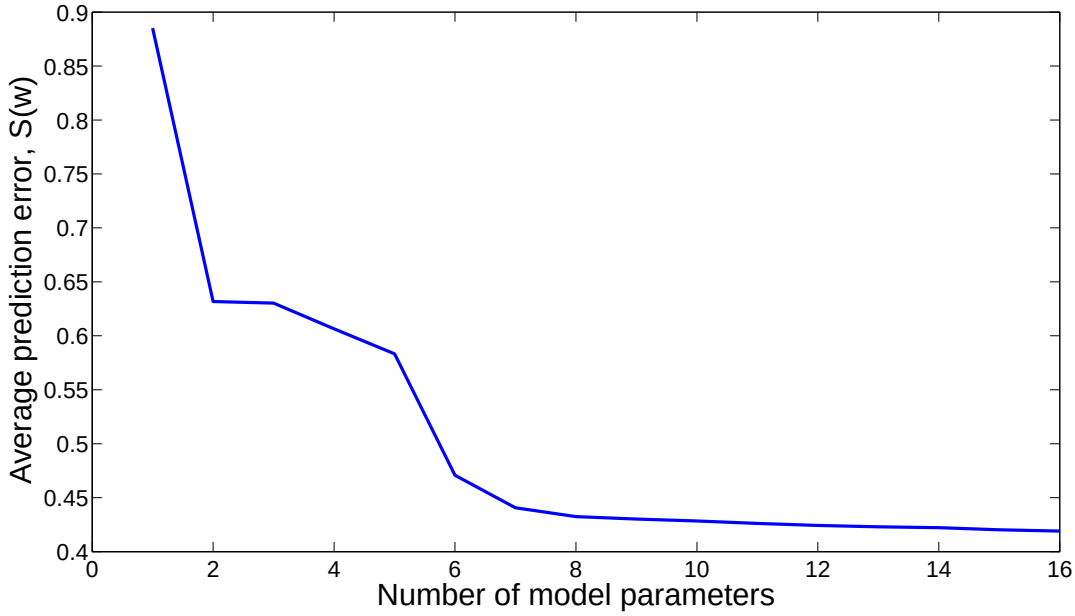

Figure 5.5: Prediction quality vs model complexity K.

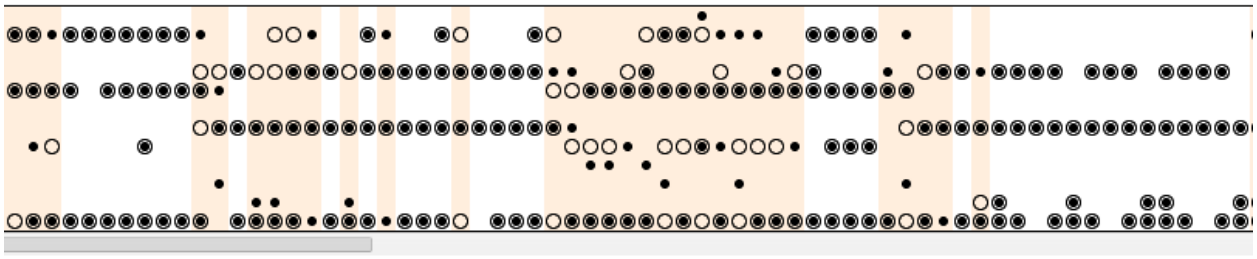

Figure 5.6: Forecasting example circles represent the truth tones, dots — predicted tones, errors are highlighted, horizontal axis is time.

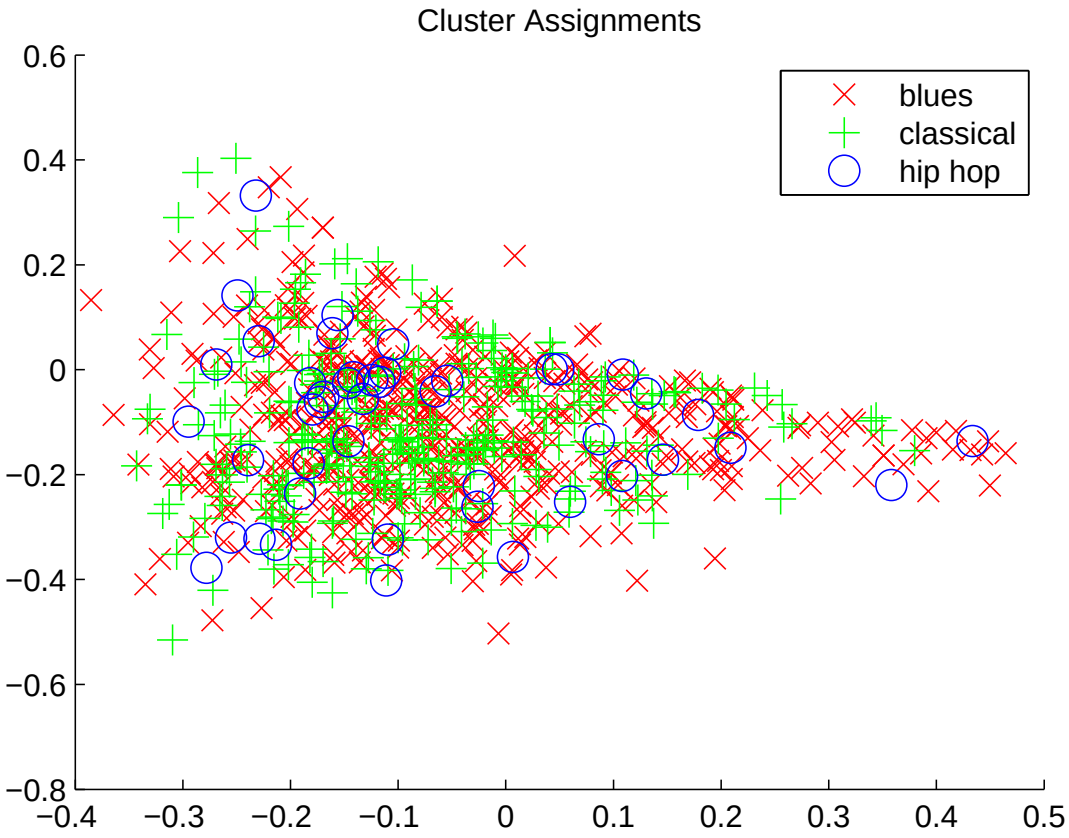

Figure 5.7: Music genres on a projection from 2-grams distribution space onto 2D space with PCA (first two components).

Appendix A

Conclusion

Overall, we managed to experiment with chords representation and to choose one, that works quite good. We tested it by developing an algorithm for predicting next element in a music sequence. We also developed several approaches to bearing with computational complexity of the problem, like smoothing error function, based on a specific distance between probability distributions; Map/Reduce-like way of gathering statistics (large scale learning). For the purpose of testing the algorithm we collected Midi50k dataset. The method was compared to other related algorithms. During tests was found that:

- The optimal model complexity (max combination length) is $K = 8$, though the more, the better.
- Number of songs in the training set should be at least 1000.
- Forecasting quality is 58% (chord-wise, 0.024% for a random guess), Hamming distance is 0.075 (meaning 92.5% tone matches comparing to 50% for a random guess).

The work can be improved by using more sophisticated classifiers and models with bigger number of parameters. It is also a great idea to include durations and arpeggio patterns and scales as a data to predicted, i. e. involving more pieces from the theory of music. Volume of different parts of the melody, as well as percussion, are also great targets. For a practical usage it can be worth creating a more interactive interface to the program, may be a graphical interface so that any user can easily apply the algorithm to predict and compose music.

Appendix B

Acknowledgments

I would like to thank everyone, who made a contribution in helping with this work and in discussing its contents. Special thanks goes to my brother, Anton Matrosov, who played a role of an expert in music theory, to Igor Evin, a major science assistant in Moscow State Conservatory, who facilitated a cross-disciplinary discussion of the proposed research, and to my academic advisor Victor Lempitsky, assistant professor at Skolkovo Institute of Science and Technology, who encouraged me to continue researching this topic.

Appendix C

Terms and abbreviations

Bayes theorem. It is one of the main theorems in the Probability theory, that allows to find a probability of some event in case another event happened, in other words, this theorem relates current probability to a prior probability. Bayes formula can be derived from basic axioms of Probability theory, in particular from conditional probability. A distinctive feature of the formula is that it requires lots of data and computations for practical usage, therefore Bayesian estimations become actively used only after advances in computational and network technologies.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}, \text{ where}$$

$P(A)$ and $P(B)$ — probabilities of A and B without regard to one other.

$P(A|B)$ — conditional probability of A given that B is true.

$P(B|A)$ — conditional probability of B given that A is true.

The statement of the theorem follows directly from the following equation:

$$P(AB) = P(A|B)P(B) = P(B|A)P(A).$$

Bayes classifier. For a pair of a vector and a class label (x, y), where $x \in \mathbf{X}$ and $y \in \mathbf{Y}$, the conditional distribution of x, given that its label y, is $P(x|y)$. Thus classification can be done as

$$C(u) = \underset{r \in \mathbf{Y}}{\operatorname{argmax}} P(y = r|x = u).$$

R, space of evaluations. The classifier outputs some ranking, that give an opportunity to distinguish between classes of objects. This valuation is an element of some set $\mathbb{R}$, it can

be set of real numbers $\mathbb{R}$, multidimensional vector of natural numbers $\mathbb{N}^d$ or any other set.

Pitch is frequency of a note. For example, the first guitar string, E_4, sounds at frequency 329.63 Hz.

Chords — set of simultaneously sounding notes. It is considered as a basic element of the musical composition. Every chord has its key (basic tone) and quality (like minor, diminished, augmented, etc). In this work we assume all the notes to be only from one octave (e.g. from C_3 to B_4).

Group of elements E is a pair {shift of the key, chord quality}, more detailed description is given in Chapter 2.1.

Music tempo. It is the speed, or pace of a given musical piece. Tempo is measured in BPM (beats per minute) and usually is between 40 and 180, depending on genre and style of some particular melody.

Probability distribution. In this work we talk about probability distributions on a finite set of chords and elements of **E**. Since that, probability distribution can be seen as a vector of real numbers from $[0, 1]$, with a unit length in L_1 metric.

Feature vector is a vector from $\mathbb{R}^d$ that somehow characterizes a specific feature of the data.

Bibliography

[1] Booth M. The AI Systems of Left 4 Dead // Valve. — 2009.

[2] Inc. Philips. Where sound meets light, comfort is built.

[3] DJ Pad, Google trends. — http://www.google.com/trends/explore#q=%22dj%20pad%22&cmpt=q&tz=. — Accessed: 2015-05-24.

[4] T. Hastie R. Tibshirani J. Friedman. The Elements of Statistical Learning. — Springer, 2001.

[5] Project repository on Sourceforge. — http://sourceforge.net/p/mlalgorithms/code/HEAD/tree/Group074/Matrosov2013MusicForecasting. — Accessed: 2015-05-24.

[6] Midi50k dataset. — https://www.dropbox.com/s/2htn17j9xduwz7u/Midi50k_full.zip. — Accessed: 2015-05-24.

[7] Mozer M. Neural network music composition by prediction // Connection Science. — 1994.

[8] D. Conklin I. Witten. Multiple viewpoint systems for music prediction // Journal of New Music Research. — 1995, rev. 2002.

[9] Brown R.G. Smoothing forecasting and prediction of discrete time series. — N.Y., 1963.

[10] Cope David. Computers and Musical Style. — Madison, 1991.

[11] Schapire Robert E. The Strength of Weak Learnability. — Boston, MA (Kluwer Academic Publishers) : Machine Learning, 1990.

[12] N. V. Filipenkov M. A. Petrova. On the analysis of multidimensional time series // JMLDA. — 2014.

[13] M. Matrosov B. Urman. Map/Reduce in application to monitoring of Worldwide LHC Computing GRID. — Dolgoprudny, Russia : Proceedings of the 56-th scientific conference of MIPT, 2013.

[14] MIDI standart, Wikipedia. — http://en.wikipedia.org/wiki/MIDI. — Accessed: 2015-05-24.

[15] A Python script to grab midi files. — http://sourceforge.net/p/mlalgorithms/code/HEAD/tree/Group074/Matrosov2013MusicForecasting/utils/midicrawler.py. — Accessed: 2015-05-24.

[16] Ken Schutte's MATLAB midi library. — http://www.kenschutte.com/midi. — Accessed: 2015-05-24.

[17] Kiwiel Krzysztof C. Convergence of approximate and incremental subgradient methods for convex optimization // SIAM Journal on Applied Mathematics 14 (3). — 2003.

[18] Marquardt D. An Algorithm for Least-Squares Estimation of Nonlinear Parameters // SIAM Journal on Applied Mathematics 11 (2). — 1963.

[19] Ch. D. Manning H. Schutze. Foundations of Statistical Natural Language Processing. — MIT Press, 1999.

[20] T. Jolliffe I. Principal Component Analysis, second edition // Springer. — 2002.

List of Figures

List of Tables