

AN INSTANCE-BASED CLASSIFICATION APPROACH TO AUTOMATIC
TRANSCRIPTION OF MONOPHONIC MELODIES


by

Fatemeh Pishdadian
A Thesis
Submitted to the
Graduate Faculty
of
George Mason University
In Partial fulfillment of
The Requirements for the Degree
of
Master of Science
Electrical Engineering


Committee:



Dr. Jill K. Nelson, Thesis Director



Dr. Kathleen E. Wage, Committee Member



Dr. Bernd-Peter Paris, Committee Member



Dr. Andre Manitus, Chairman, Department
of Electrical and Computer Engineering



Dr. Kenneth S. Ball, Dean, Volgenau School
of Engineering

Date: _____

Fall 2013
George Mason University
Fairfax, VA

An Instance-Based Classification Approach to Automatic Transcription of Monophonic
Melodies

A thesis submitted in partial fulfillment of the requirements for the degree of
Master of Science at George Mason University

By

Fatemeh Pishdadian
Bachelor of Science
Ferdowsi University of Mashhad, 2005

Director: Dr. Jill K. Nelson, Associate Professor
Department of Electrical and Computer Engineering

Spring Semester 2014
George Mason University
Fairfax, VA

Copyright © 2014 by Fatemeh Pishdadian
All Rights Reserved

Dedication

To Maman, Baba, and Hamid

Acknowledgments

I would like to give special thanks to my advisor, Dr. Jill Nelson, for her help and guidance throughout writing this thesis. I would also like to express my gratitude to the members of my thesis committee, Dr. Kathleen Wage and Dr. Bernd-Peter Paris, for their invaluable comments and their constant support.

Table of Contents

	Page
List of Tables	vii
List of Figures	ix
Abstract	xii
1 Introduction	1
2 Background and Prior Work	6
3 Monophonic Transcription Algorithm	10
3.1 Algorithm overview	10
3.2 Time-domain feature extraction	11
3.2.1 Piano sound signal in time domain	11
3.2.2 Onset and sustain instances	11
3.3 Frequency-domain feature extraction	12
3.3.1 Spectrogram - Time/Frequency parameter setting	12
3.3.2 Dominant-octave detection	13
3.3.3 Feature vector extraction	17
3.4 Pitch detection	17
3.4.1 K-Nearest Neighbor algorithm	17
3.4.2 Semi-classification-based approach	20
3.4.3 Alternative distance measures	20
3.4.4 Spectrum normalization	22
3.4.5 Training database	24
3.4.6 Testing database	24
3.5 Note sequence detection	25
3.5.1 Viterbi algorithm	27
3.5.2 Stack-based tree search	28
3.5.3 Path metric derivation	30
4 Results and Discussion	38
4.1 Performance evaluation	38
4.1.1 Conventional KNN algorithm	38
4.1.2 Two step algorithm (semi-KNN plus sequence tracking)	41

4.2	Computational complexity	49
4.2.1	K-Nearest Neighbor algorithm	49
4.2.2	Note sequence tracking	49
4.2.3	The one-step algorithm versus the two-step algorithm	51
5	Conclusion	53
	Bibliography	55

List of Tables

Table	Page	
3.1	Fundamental frequency range of piano octaves - the frequency components above 4186 Hz are assigned to the 9 th region.	14
3.2	Numerical values of the spectrogram parameters. The window length is adjusted so that the fundamental frequencies of the two lowest notes of the dominant octave can be resolved.	17
3.3	Updated stacks in the example of stack-based tree search.	29
4.1	Note classification performance accuracy for the conventional K-Nearest Neighbor algorithm over two different datasets, and with different number of neighbors included in the voting procedure. Euclidean distance measure is used in all the experiments.	39
4.2	Note classification performance accuracy for the conventional K-Nearest Neighbor algorithm over two different datasets, and with different number of neighbors included in the voting procedure. City block distance measure is used in all the experiments.	40
4.3	Note classification performance accuracy for the conventional K-Nearest Neighbor algorithm over two different datasets, and with different number of neighbors included in the voting procedure. Correlation distance measure is used in all the experiments.	41
4.4	Note classification performance accuracy for the validation and testing sets, both separately and combined. The Viterbi algorithm and stack-based tree search (with and without a naive bias term ($\alpha = \frac{1}{88}$)) are employed at the note sequence tracking stage.	42
4.5	Computational complexity of the Viterbi algorithm and the stack-based tree search in terms of average number of trellis or tree transitions (or metric computations) per testing sample (theme). Testing samples are 22 notes long on average.	51

4.6	Computational complexity of the two-step algorithm combining distance calculations in the candidate selection stage and metric calculations in the sequence tracking stage.	52
-----	---	----

List of Figures

Figure	Page
1.1 Block-diagram of a full bottom-up AMT system. In the bottom-up approach properties of individual notes are extracted prior to piece-related information such as tempo, key signature, and expression marks.	3
3.1 Block diagram of two different melody transcription methods. The conventional KNN algorithm is employed provided that the training database is sufficiently large. If the database is of minimum size, a note sequence tracker is combined with a semi-KNN-based pitch candidate selector to compensate for the shortage of data by incorporating musicological information in the transcription process.	10
3.2 ADSR Model for piano sound signal.	12
3.3 Spectrogram divided into frequency range of piano octaves. The frequency axis is logarithmically scaled.	16
3.4 The 4-Nearest Neighbor classifier labels a query sample according to the majority vote among its four nearest neighbors in the training database. In this example, three out of four nearest neighbors have positive labels, therefore, S_q is classified as positive.	19
3.5 The 1-Nearest Neighbor classifier assigns to a query sample the label of its nearest neighbor among training samples. In this example, the nearest neighbor to the query sample has a negative label, therefore, S_q is classified as negative.	19
3.6 Illustration of the unit circle in 1-norm (city block distance measure). . . .	21
3.7 Illustration of the unit circle in 2-norm (Euclidean distance measure). . . .	21
3.8 Recorded piano tone A4 (with fundamental frequency equal to 440 Hz) in time domain.	25
3.9 Training feature vector from pitch class 69 (normalized to the maximum value).	25
3.10 Spectrum of the note D2 (MIDI number 18), $F_0 = 73.4$ Hz.	26
3.11 Spectrum of the note D3 (MIDI number 30), $F_0 = 146.8$ Hz.	26

3.12	(1) Trellis structure for a state sequence of length four, and two states at each time step. $s_{\tilde{k}l}$ denotes the \tilde{k}^{th} state at time step l , and the numbers next to arrows are transition lengths. (2) Output of the Viterbi algorithm - the sequence of states with minimum length.	28
3.13	Progress of stack-based tree search in a simple case.	29
3.14	Deviation of the observed spectrum from pitch candidate spectra, modeled with Gaussian distributions.	33
3.15	Gaussian weights assigned to training feature vectors according to their distance from the observed feature vector.	34
3.16	Note transition probability distribution (bigram). The values on the x-axis indicate the transition distance in semitones (half notes) and on the y-axis the corresponding probabilities. For example, the probability of moving up one half step between two consecutive notes is roughly 0.1. In our algorithm the bigram is computed empirically from J. S. Bachs Inventions No. 1, 2, and 3.	36
4.1	Error rate distribution over piano notes for the two-step algorithm with one candidate (technically the 1NN algorithm with the minimum-size database and Euclidean distance measure).	44
4.2	Error rate distribution over piano notes for the two-step algorithm with two candidates. Note sequence tracking is carried out via the Viterbi algorithm.	44
4.3	Output of the conventional KNN algorithm for a theme of length 26 notes. Correlation distance measure along with the large database (10 samples per class) are employed. With $K = 3$, three notes are misclassified, while setting $K = 5$ increases the number of misclassifications to five.	45
4.4	Output of the two-step algorithm with one and two candidates for a theme of length 23 notes. In this example adding the second candidate corrects all the three classification errors made using only one candidate.	46
4.5	Output of the two-step algorithm with one and two candidates for a theme of length 23 notes. In this example adding the second candidate corrects the majority of misclassifications (7 out of 8) but not all of them.	47
4.6	Output of the two-step algorithm with one and two candidates for a theme of length 23 notes. Adding the second candidate degrades the performance of the system from three misclassifications to five.	48

4.7	Output of the two-step algorithm for the same example as in Figure 4.6. Adding the third candidate fixed three of five misclassifications resulting in 21 correctly classified notes out of 23.	48
-----	---	----

Abstract

AN INSTANCE-BASED CLASSIFICATION APPROACH TO AUTOMATIC TRANSCRIPTION OF MONOPHONIC MELODIES

Fatemeh Pishdadian, M.S.

George Mason University, 2014

Thesis Director: Dr. Jill K. Nelson

Automatic music transcription (AMT) is a relatively new application in the field of music signal processing. The purpose of an AMT algorithm is to transform a raw acoustic musical signal into a written version, namely a score. The most basic pieces of information an AMT system aims to extract from a raw acoustic musical signal are the properties of individual note events, such as the starting time (onset), duration, and pitch. Because of its overwhelming complexity, the transcription problem has been broken down into sub-tasks, and separate algorithms have been developed over the years to address different operations in the overall system. Pitch detection is an important part of any transcription system, and has been the subject of a vast volume of research over the past two decades.

Estimation of a single pitch at each time step is known as monophonic pitch detection. In this work, we present an instance-based classification approach to transcription of monophonic melodies. Depending on the size of training database, two different pitch classification methods are proposed. The conventional K-Nearest Neighbor algorithm is trained on a large database of piano notes and employed for pitch detection.

A two-step algorithm, combining semi-KNN pitch candidate selection and note sequence tracking is suggested to deal with cases in which the training database is of minimum size, containing one sample per class. It is demonstrated that in the abundance of training data, the KNN algorithm along with a proper choice of the distance measure and K , yields high performance accuracy. Furthermore, while maintaining low computational complexity, the proposed two-step algorithm is capable of compensating for the shortage of data by incorporating prior musicological information in the transcription process.

We note that monophonic pitch detection is a mature problem compared to polyphonic pitch detection, which is the main focus of current studies. Nevertheless, monophonic pitch detection can still be of interest since a considerable portion of music corpora is composed of single line melodies. One of the shortcomings of the available monophonic algorithms, which are mostly based on signal processing techniques such as autocorrelation function (ACF) or spectral peak picking, is that they are under-evaluated in terms of frequency range and melodic structure.

Classification-based pitch detection algorithms have been proposed later on and particularly developed for polyphony. Despite their promising performance accuracy, the classification methods that have been employed to solve the multi-pitch detection problem, namely Support Vector Machines (SVMs) and Artificial Neural Networks (ANNs), are computationally too demanding to be utilized for the monophonic case which is a simpler scenario. The work presented in this thesis was motivated by a need for monophonic transcription techniques with significantly reduced training time, low run-time complexity, and the capability to explore melodic contours.

Chapter 1: Introduction

Music transcription refers to the process of transforming a raw acoustic musical signal into a written symbolic representation, namely a score. Such a task, if performed by a computer rather than by a human, would naturally be called automatic music transcription. The need for Automatic Music Transcription (AMT) systems has been increasing during the past decade. What has primarily given rise to this need is the rapid growth of digital music databases, implying the necessity for novel archiving and searching methods.

For designing musical search engines with efficient performance on large databases, the same principle employed in full-text search engines has been proposed and applied in previous work. Full-text search engines work based on excerpts of text specified by a user, which distinguishes them from searching methods based only on bibliographical information such as title, author, etc. A musical search engine can similarly examine the available songs or whole pieces of instrumental music in the database to find a match for a provided excerpt of music, be it a hummed melody, tapped rhythmic pattern, or a short recording from a live performance. Nevertheless, a raw musical excerpt, whatever the content, does not provide the search engine with appropriate tools. Useful information for initializing a search needs to be extracted from the input signal. In effect, a transcription method can be exploited to retrieve required information from the input data.

Conventional music transcription (by musicians), although still superior in terms of accuracy to currently available automatic methods, is a specialized task requiring many hours of training. The way a human transcriber approaches the task can be regarded as a *top-down* or *rule-based* mechanism, since the human brain tends to analyze music in the context of prior information including form, genre, tonality, etc. On the other hand, computer algorithms are capable of extracting low-level information, e.g. instantaneous frequency, directly from an acoustic signal. Initializing the transcription task by low-level

information, a so called *bottom-up* mechanism, is a skill humans normally do not possess. In addition, automatic approaches offer a considerable amount of speed and computational power, which makes them better suited to real-time applications. If provided with structured musicological information, automatic methods can also combine the top-down and bottom-up mechanisms in order to achieve better overall performance. Because of the mentioned promising characteristics, the development of AMT systems has attracted much attention in the field of music signal processing.

Musipedia (<http://www.musipedia.org>) and Muma [1] are examples of music search engines working based on extracted score elements from an input musical signal. Other applications of AMT systems include automatic instrument tutoring, where a student's performance is evaluated by making a comparison between the transcribed version of the performance and the original score [2] [3], as well as musicological analysis, where the subject of study is either artistic (expressive) deviation of a performance from the tempo, dynamics, and articulation specified in the score, or undocumented (e.g. improvised or ethnic) music.

The block-diagram of a full bottom-up AMT system is depicted in Figure 1.1. Ideally, the input to the system is a raw acoustic musical signal and the output a fully annotated score. The system extracts information from the musical signal in a hierarchical manner. In the lowest stage, the most basic information, i.e., the properties of individual notes such as the onset, duration, and pitch are extracted. Higher level information – namely the key, rhythm, melodic structure, etc. – forms the output of the following stage. In the highest level, the system adds to the score the expression marks which altogether serve as a performance guide in terms of articulation, sound level dynamics, tempo variations, etc.

Due to the nonstationarity of musical signals in general, automatic transcription has proved to be a challenging task. To gain some perspective on the temporal dynamics of music, consider a piece with a fairly moderate tempo, e.g. 80 bpm (beats per minute), and 4/4 time signature. The beat reference in this simple case is a quarter note, which according to the tempo measure lasts only 0.75 seconds. The eighth and sixteenth notes, two subsequent note divisions, have a duration of 0.375 and 0.1875 seconds respectively.

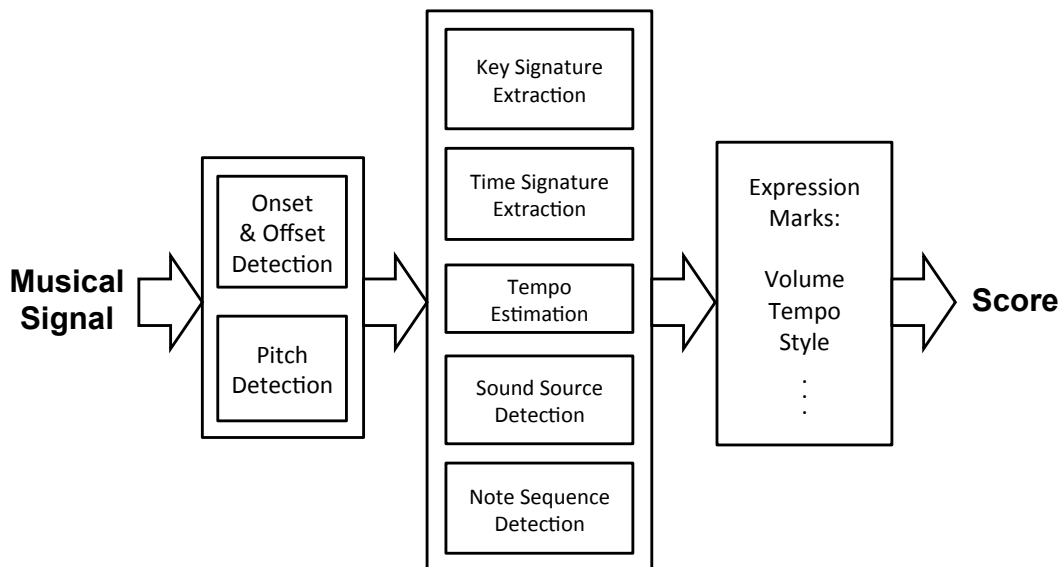


Figure 1.1: Block-diagram of a full bottom-up AMT system. In the bottom-up approach properties of individual notes are extracted prior to piece-related information such as tempo, key signature, and expression marks.

Thus, a change in the properties of the signal, for instance a shift in the pitch of an ongoing sound or arrival of a single or group of notes, would be expected every half a second on average. A good transcription algorithm should therefore be robust enough to make high accuracy estimates based on limited observation.

Harmonicities, an inherent property of musical sound, is another factor rendering automatic transcription challenging. What the human brain perceives as a single tone, called *single pitch* in music terminology, is in fact a combination of a fundamental frequency (f_0) and its related harmonics or *overtones* ($2f_0, 3f_0, \dots$). The fundamental frequency of a typical musical sound is most often its strongest component, in other words, the power content of the signal is mainly concentrated around the fundamental frequency. Making this general assumption, magnitude thresholding could be suggested as the most intuitive method for (single) pitch estimation. However, noise or acoustical properties of an instrument might attenuate the first harmonic more than higher harmonics, resulting in selection of incorrect peaks in the thresholding procedure. Power spectra of tones in the lowest register of the

piano are examples of such a case. Octave error, where the second harmonic ($2f_0$) is returned as the fundamental frequency, is the most common pitch detection error caused by weakened first harmonic.

Pitch detection can also be challenging because of overlapping harmonics of notes in different registers. Assuming the well-known twelve-tone system, frequencies of the second and third harmonics of a note in a given octave are equal to the fundamental frequencies of the first and eighth notes in the next octave. For instance, note C_3 has a fundamental frequency of around 130.8 Hz, and its second and third harmonic frequencies are around 261.6 Hz and 392 Hz, which happen to be the fundamental frequencies of notes C_4 and G_4 respectively. In an even more complicated scenario, multiple notes could be played at the same time (polyphony). Two phenomena might occur in the signal spectrum: magnitudes of perfectly overlapping harmonics would sum giving rise to a distinct stronger harmonic, or intermodulation between harmonics at slightly deviated frequencies would generate a spectral component with a wider lobe, and thus, lower frequency resolution. In either case, detection of single pitches given the superimposed spectra is a nontrivial task.

Time-domain characteristics of a musical signal are yet another factor to be taken into account in the transcription process. For instance, based upon the sound producing mechanism, the shape of the time-domain envelope and its power content at the start-time of a note, known as *onset*, differ from one instrument to another. The sound produced by hammered string (piano) or plucked string (guitar) instruments features an impulsive onset, while the onset of notes generated by bowed string (violin) or wind (flute) instruments displays a smoother behavior. Naturally, a transcription algorithm must be capable of detecting onsets of notes played by either group of instruments, individually and combined.

Because of above mentioned characteristics of musical data, a purely bottom-up transcription method, which solely relies on acoustical information would be likely to yield erroneous results. Besides, such an approach treats music from different epochs, forms, and genres exactly the same and merely at the physical level. As mentioned earlier, the performance of an AMT system could be improved by taking advantage of prior information

about the type of music under study, which is analogous to the approach taken by a human transcriber. A large portion of music corpora (almost the entire collection of western music dated prior to the 20th century) is composed based on well-established musicological rules. These rules can be incorporated in the transcription procedure in the form of probabilistic models for melodic and harmonic structures.

Although all transcription subtasks depicted in Figure 1.1 are equally important, because of the complexity of the problem, in this work we mainly focus on pitch detection and note sequence tracking stages. Moreover, we limit our database to monophonic melody lines played by piano. A modern pianoforte covers fundamental frequency range of 27.5 Hz - 4186 Hz, or equivalently 88 note pitches in seven full and two incomplete octaves. Since this is a broad frequency (pitch) range compared to that of most musical instruments, we anticipate that a pitch-detection algorithm developed for piano could be generalized to other instruments with slight modification.

The thesis is organized as follows: A review of the existing monophonic transcription algorithms is presented in Chapter 2. In Chapter 3, we present our approach to transcription of monophonic melody lines and describe each part of our algorithm in detail. Experimental results are provided and discussed in Chapter 4. The work is concluded and some of the open questions are laid out in Chapter 5.

Chapter 2: Background and Prior Work

Monophonic transcription is a relatively mature problem within music signal processing and has been studied by the research community for nearly four decades. Although the current research is primarily focused on the polyphonic transcription problem, monophonic transcription is still of interest. As a matter of fact, single melody lines form a portion of music corpora which is by no means negligible. Single voice chants and ethnic instrumental performances are two examples of monophonic melodies. Moreover, current online *query by humming* systems initiate their search based on a short recording of a hummed single line melody [4].

One of the earliest monophonic transcription methods, proposed by Piszczalski and Galler [5], tracks note pitches through amplitude thresholding in the frequency domain. Such an approach yields meaningful results as long as the fundamental frequency remains the strongest component of the musical sound. For this reason, the method was restrictively tested on the sound produced by recorder and symphonic flute.

Conventional single pitch tracking algorithms were originally developed for speech processing purposes, e.g. for speaker identification and voice classification [6]. These early methods can be categorized according to their analysis domain [7]. Time-domain pitch tracking approaches include the use of Zero-crossing rate (ZCR), autocorrelation function (ACF) [8] [9] [10], the YIN algorithm [11] (a modified time-domain autocorrelation method), average magnitude distance function (ADMF), cross-correlation function (CCF), etc., for estimating the fundamental frequency, given the time envelope of an acoustic signal. Some of the frequency-domain methods include cepstrum analysis, spectrum autocorrelation, harmonic matching [12], and cross-correlation with the ideal timbral pattern (matched filter-based approach) [13].

One of the primary shortcomings of the existing work on monophonic transcription, which motivates the revision of the problem, is that most of the algorithms are insufficiently evaluated. They are tested either over the sequential order of notes within a scale [14] [13] [15] or on a very limited range of frequencies [16]. Scales and narrow frequency ranges are hardly representative of actual melodic structures, and hence more sophisticated structures must be considered for meaningful algorithm evaluation.

In [17] pitch tracking methods are divided into two groups called *spectral place models* and *spectral interval models*. Spectral place models attempt to estimate the fundamental frequency from the location of harmonics, whereas spectral interval models take the interval between spectral peaks into account in order to find the pitch. All of the aforementioned methods fall into a broader category, which is usually referred to as signal processing-based pitch tracking (as opposed to classification-based pitch tracking). The interested reader may refer to [18] and [19] for a rather detailed summary of signal processing-based pitch detection methods.

Classification-based pitch detection methods have been proposed later than signal processing-based methods and to our knowledge never applied to single voice melodies. These approaches, mainly proposed for polyphonic music, have demonstrated promising results. One example is the partial (harmonic) tracking technique based on the combination of adaptive oscillators and artificial neural networks (ANNs) proposed by Marolt in [20]. Adaptive oscillators are systems with frequency and phase as internal parameters. Given a periodic signal as the input, the oscillator can adjust its phase and frequency to match that of the signal. Thus, a network of oscillators can be used to detect the partials contained in a musical signal. In [20], 76 neural networks are trained on piano tones *A1* to *C8* (all the keys except the lowest octave). The input to each NN is a collection of partials detected by the network of oscillators, and the output a single value. If the output value passes a certain threshold, it indicates the presence of the target note.

Another example of classification-based pitch detection methods is the use of Support Vector Machines (SVMs) for modeling piano tones suggested by Poliner and Ellis in [21] and

[22]. SVM is a supervised binary classification algorithm that represents training samples as points in a high-dimensional feature space. The name one-versus-all (OVA) has also been used to indicate the binary classification procedure, that is, deciding whether or not a new sample belongs to a certain training class. In particular, hyperplanes separating different classes are found in an optimization scenario such that the gap between training classes is maximized. Query samples are mapped into the same space and classified depending on their location with respect to the hyperplane (class 1 if it lies on one side, and -1 if on the other side) [23]. In [22], Poliner et al. trained 87 OVA support vector machines to detect each piano tone in a short piece of polyphonic piano music.

It should, however, be noted that polyphonic pitch detection is a more complicated problem than monophonic pitch detection. In order to address the challenges of the multi-pitch estimation task, therefore, classification methods with a high level of complexity, such as the two examples mentioned earlier, have been proposed in the literature. Even though the performance of the selected pitch classifiers has proven to be satisfactory in the polyphonic scenario, they require a long and computationally demanding training stage. Consequently, solving the monophonic transcription problem via these complex methods would be computationally inefficient.

The motivation for this work is to develop a classification-based monophonic transcription technique with significantly reduced training time, low run-time complexity, and the capability to explore melodic contours. The pitch detection in our approach is performed via the K-Nearest Neighbor (KNN) algorithm trained on piano tones. K-Nearest Neighbor is a low complexity method with a very simple training stage. To our knowledge, KNN has not been applied to pitch detection in the literature. Our experiments show that, when provided with sufficient training data, KNN can yield high performance accuracy. In the case of a very small training set, KNN can be combined with a sequential detection step to improve pitch sequence transcription. The combination of two steps still imposes a low computational burden compared to the aforementioned classification-based methods, which

are quite popular in polyphonic transcription. In the design of our sequence tracking algorithm, we have been inspired by the probabilistic approach taken in [24], which employs an empirical note transition probability distribution for note sequence estimation.

Chapter 3: Monophonic Transcription Algorithm

3.1 Algorithm overview

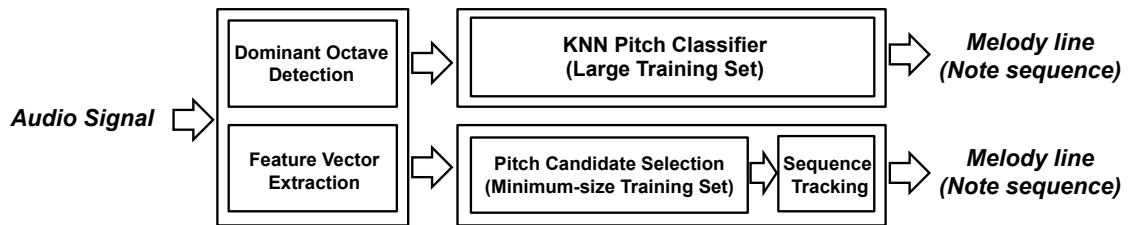


Figure 3.1: Block diagram of two different melody transcription methods. The conventional KNN algorithm is employed provided that the training database is sufficiently large. If the database is of minimum size, a note sequence tracker is combined with a semi-KNN-based pitch candidate selector to compensate for the shortage of data by incorporating musicological information in the transcription process.

A block diagram of the two proposed transcription algorithms is provided in Figure 3.1. Both methods share a feature extraction stage. In the feature extraction stage, the dominant octave, where the majority of the notes in a melody line are played, is determined. Moreover, frequency-domain features of note events are extracted from the input audio signal according to time-domain labels (see Section 3.2). If a sufficiently large training database is available, the conventional KNN algorithm (upper path) utilizes the extracted feature vectors to identify the target pitch class for each note event. In the case of a minimum size database containing only one sample from each class, the transcription is performed via a two-step algorithm in order to improve the transcription accuracy. In the first sub-block of the lower path, a semi-KNN approach is employed to identify pitch candidates for a given note event.

In the second sub-block, the sequential relationship between the pitch candidates is explored and the most likely note sequence or *melody line* is returned as the system output. Different parts of the system are described in more detail in the following sections of this chapter.

3.2 Time-domain feature extraction

3.2.1 Piano sound signal in time domain

The time-domain envelope of a piano note event can be divided into four distinct sections: attack, decay, sustain, and release (ADSR model [25] illustrated in Figure 3.2). A note event physically starts when the hammer hits the string and ends when the string stops vibrating. During the first few milliseconds of a piano tone, a sudden burst of energy that contains a broad range of frequencies is observed. This section, which shows an almost impulsive behavior, is called the attack part and is modeled as a rising exponential function with a very small time constant. The decay part, during which the signal level drops rapidly, could also be modeled by an exponential function, though with a larger time constant compared to the attack part. The decay section is followed by the sustain section, lasting tens of milliseconds, during which the signal maintains a stable energy level as well as a fairly stable pitch. Our definition of a stable pitch is a combination of the note fundamental frequency and its related harmonics which are the prominent components in the signal spectrum for a sufficiently long period. In the release part, all frequency components fade out and the signal level gradually falls to zero.

3.2.2 Onset and sustain instances

As mentioned in the previous section, the starting time of a piano note event is best characterized by the impulsive behavior of the envelope during the attack time and is more specifically called a *hard onset* (as opposed to the term *soft onset* used in the case of instruments like violin, where the arrival of a new note takes place more gently and such outburst of energy is not observed).

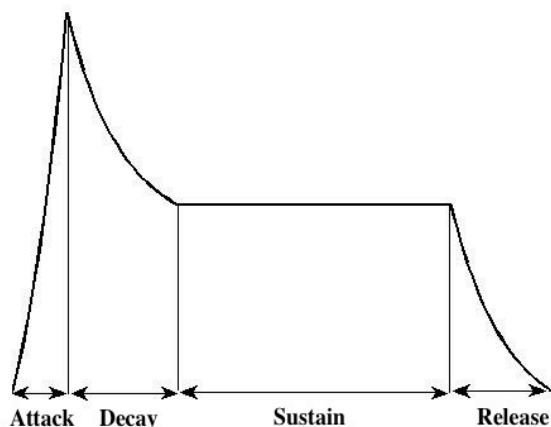


Figure 3.2: ADSR Model for piano sound signal.

The sustain part of a note is another important time-domain feature to be extracted. Since the note pitch is more stable during the sustain part, it is preferred over other parts to be employed in the pitch detection stage. We note however that automatic detection of onset and sustain instances are outside the scope of this work. In order to prevent the erroneous results of any sustain detection method from having an impact on the performance of our algorithm, we label the sustain instances manually.

3.3 Frequency-domain feature extraction

3.3.1 Spectrogram - Time/Frequency parameter setting

The spectrogram provides a powerful tool for joint time-frequency analysis of an audio signal. It is generated by windowing the time-domain signal and computing the Short-Time Fourier Transform (STFT) at each time frame. The window type, window length and hop size between successive frames are among adjustable parameters controlling temporal and spectral features such as time resolution, frequency resolution and spectral leakage. In our system, a Hamming window is employed, whose length and hop size is determined based on the output of the dominant octave detection stage. A detailed explanation of the

dominant octave detection method will be presented in Section 3.3.2.

Let us assume that the $I \times J$ matrix P is the spectrogram of a given musical signal, $x[n]$ ($n = 0, \dots, N - 1$). Then the column vector number j in P contains the power spectral density of the j^{th} windowed segment of $x[n]$, denoted by $x_{wj}[n]$,

$$x_{wj}[n] = x[n]w[n - r(j - 1)] \quad j = 1, \dots, J, \quad (3.1)$$

where r denotes the hop size. That is,

$$P_j = \frac{2}{f_s \|w[n]\|_2^2} |X_{wj}[k]|^2 \quad j = 1, \dots, J, \quad (3.2)$$

where $\|\cdot\|_2$ indicates the 2-norm of a vector, $X_{wj}[k]$ is the κ -point discrete Fourier transform of $x_{wj}[n]$, and κ is taken to be sufficiently large to prevent time-domain aliasing. The power spectral density is normalized inversely by the window norm to yield a constant level of power per frequency bin independent of the window length [26].

3.3.2 Dominant-octave detection

Fundamental frequencies of musical notes in western music (equal temperament) are distributed on a logarithmic scale. That is, lower fundamental frequencies are more closely spaced than higher ones. Therefore, higher frequency resolution is required for resolving two adjacent notes in lower registers. In the frequency range of piano notes, the smallest and largest spacing between note fundamental frequencies are around 1.64 Hz (between A_0 and $A_0^\#$), and 234.94 Hz (between B_7 and C_8) respectively.

The frequency resolution of the spectrogram is basically determined by the main lobe width of the window. Although, the half main-lobe width could still be a meaningful metric for measuring the frequency resolution. The approximate main-lobe width of the Hamming window used in our algorithm can be calculated as $\frac{8\pi}{M}$, where $M + 1$ is the window length in number of samples. The window length that results in a frequency resolution equal or

higher than 1.64 Hz would be at least 53780 samples with sampling rate of 44.1 kHz, or equivalently 1.2195 seconds. Given the non stationary nature of the musical signal, this window is too long to capture the time-domain characteristics efficiently.

In order to deal with the inherent trade-off between time and frequency resolutions, we suggest an adaptive resolution setup (multi-resolution approach) based on the dominant frequency range of the notes in a given piece of music. In this method, first we obtain a preliminary version of the spectrogram with high time resolution (window length of around 20 ms and 50% overlap between successive frames) for the given piece. The spectrogram is divided into 9 regions, each roughly corresponding to the fundamental frequency range of one piano octave (7 full and two incomplete octaves, see Table. 3.1). At each frame, the octave frequency band containing the highest amount of power is found and recorded as the short-term dominant octave. The dominant octave for the whole length of a piece is determined based on the majority vote among all the short-term dominant octaves.

Table 3.1: Fundamental frequency range of piano octaves - the frequency components above 4186 Hz are assigned to the 9th region.

Octave Number	Fundamental Frequency Range (Hz)
0	27.5 - 30.9
1	32.7 - 61.7
2	65.4 - 123.5
3	130.8 - 246.9
4	261.6 - 493.9
5	523.3 - 987.8
6	1046.5 - 1975.5
7	2093 - 3951.1
8	> 4186

Let e_m denote the range of elements in the j^{th} column of the spectrogram given by (3.2), whose corresponding frequency components lie within the frequency range of the m^{th} piano octave ($m = 0, \dots, 8$). The spectral power content in the frequency range of the m^{th} octave in the j^{th} window frame can thus be obtained from

$$P_{mj} = \sum_{i \in e_m} P_{ij} \quad j = 1, \dots, J \quad m = 0, \dots, 8. \quad (3.3)$$

The following expressions describe the computation of the dominant octave

$$m_j^* = \underset{m}{\operatorname{argmax}} P_{mj}, \quad (3.4)$$

$$M^* = \underset{j}{\operatorname{MV}} \{m_j^*\}, \quad (3.5)$$

where m_j^* and M^* indicate the short-term dominant octave and the total dominant octave respectively, and $\underset{j}{\operatorname{MV}}\{.\}$ ($j = 1, \dots, J$) is the *majority vote* operation over all the spectrogram columns.

With this adaptive approach, the time resolution is only compromised when it is absolutely necessary to have a certain level of frequency resolution to resolve the fundamental frequency of the majority of notes in a piece of music. Furthermore, better results are expected when we are dealing with pieces whose notes mostly reside in lower octaves. Figure 3.3 depicts the octave-wise segmented spectrogram for an excerpt of music of around 11 seconds.

It should be noted that the dominant octave cannot be estimated by merely considering the frequency-domain representation of the whole piece. As explained in Section 3.2.1, the total power of a piano note event is not evenly distributed over time. For instance, the attack part is distinguished by its impulsive behavior carrying a large amount of energy in a very short time, while the sustain part maintains a moderate energy level though for a

longer time.

On the other hand, the power content of the attack part is spread over a broad range of frequency components as opposed to the sustain part whose power content tends to be mainly concentrated around the fundamental frequency and therefore within the frequency range of a particular octave. Since the power content of the sustain part provides more useful information for estimating the dominant octave, it must be given more weight in our computations. The frequency-domain representation of the signal combines the power content of the whole sound signal. Thus, it is impossible to determine to which part of the envelope the power content belongs. By employing a high time-resolution spectrogram not only the power content of the envelope parts can be easily separated, but also weighting becomes a natural part of the process. That is, the sustain section which is longer than attack section falls within the range of many more time frames and hence its vote for the dominant octave is counted many more times. Numerical details of the spectrogram parameter setting in a multi-resolution scenario are presented in Table 3.2.

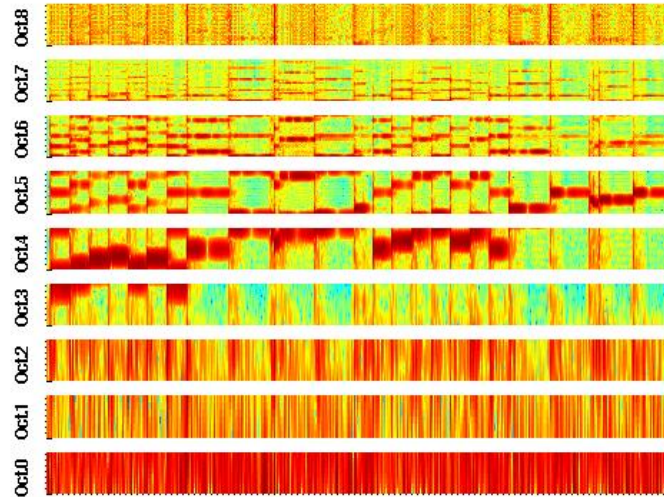


Figure 3.3: Spectrogram divided into frequency range of piano octaves. The frequency axis is logarithmically scaled.

Table 3.2: Numerical values of the spectrogram parameters. The window length is adjusted so that the fundamental frequencies of the two lowest notes of the dominant octave can be resolved.

Dominant Oct.	Oct.0	Oct.1	Oct.2	Oct.3	Oct.4	Oct.5	Oct.6	Oct.7	Oct.8
Win. Length (s)	1.25	1.16	0.56	0.28	0.16	0.14	0.07	0.05	0.02
Hop Size (ms)	29.39	27.21	13.06	6.53	3.81	3.26	1.63	1.09	0.54
ΔF (Hz)	1.59	1.72	3.59	7.18	12.3	14.36	28.71	43.07	86.13

3.3.3 Feature vector extraction

While non-stationary in general, the musical signal could be regarded as stationary if only a very short segment of it is under study. Columns of the spectrogram are in effect frequency-domain representations (periodograms) of such short segments. In particular, columns extracted from the sustain part of a note carry significant information about main spectral components required for pitch detection. In our system, these columns serve as frequency-domain feature vectors forming the pitch classification dataset. In order to reduce the spectrum estimation variance, the Welch method [27] (windowing the signal and averaging the periodograms) is applied by simply extracting several consecutive columns from the sustain part of each note event and taking their average. The number of extracted columns depends on the duration of a note, as well as the time resolution. It has to be large enough to reasonably reduce the spectrum estimation variance, but not too large to include columns from any part of the note other than the sustain part.

3.4 Pitch detection

3.4.1 K-Nearest Neighbor algorithm

K-Nearest Neighbor (KNN) is a well known instance-based classification algorithm whose training stage consists only of collecting training samples [28]. In this regard, the algorithm

offers a high level of simplicity compared to other adopted pitch classification methods such as using Support Vector Machines (SVMs) to model the piano tones.

The KNN algorithm works based on the assumption that each training or testing sample corresponds to a point in the n -dimensional Euclidean space, \mathbb{R}^n . Thus, an arbitrary sample could be described by a feature vector of length n , i.e,

$$S = [S(1), S(2), \dots, S(n)]^T, \quad (3.6)$$

with S and $S(n)$ denoting a column vector and the value of its n^{th} attribute respectively. The Euclidean distance between two samples, $S^{(i)}$ and $S^{(j)}$, is then defined as

$$d_{Euc}(S^{(i)}, S^{(j)}) \equiv \sqrt{\sum_{h=1}^n (S^{(i)}(h) - S^{(j)}(h))^2}, \quad (3.7)$$

which can alternatively be stated in terms of the 2-norm of the difference between feature vectors:

$$d_{Euc}(S^{(i)}, S^{(j)}) \equiv \|S^{(i)} - S^{(j)}\|_2. \quad (3.8)$$

Using (3.7) or (3.8), the algorithm computes the distance of a given query (testing) sample from all or a subset of training instances. The algorithm which takes all the training instances into account is called the *global* method; as opposed to the *local* method, where only the nearest training instances are considered. In the following step, the query (testing) sample is assigned to the most common training label among K nearest neighboring points. For instance, the 4-Nearest Neighbor classifier shown in Figure 3.4 classifies the testing sample as positive, since three out of four nearest neighboring points have positive labels. It should be clear that when $K = 1$, the testing sample will be labeled with the same target value as its closest neighbor. This case is illustrated in Figure 3.5.

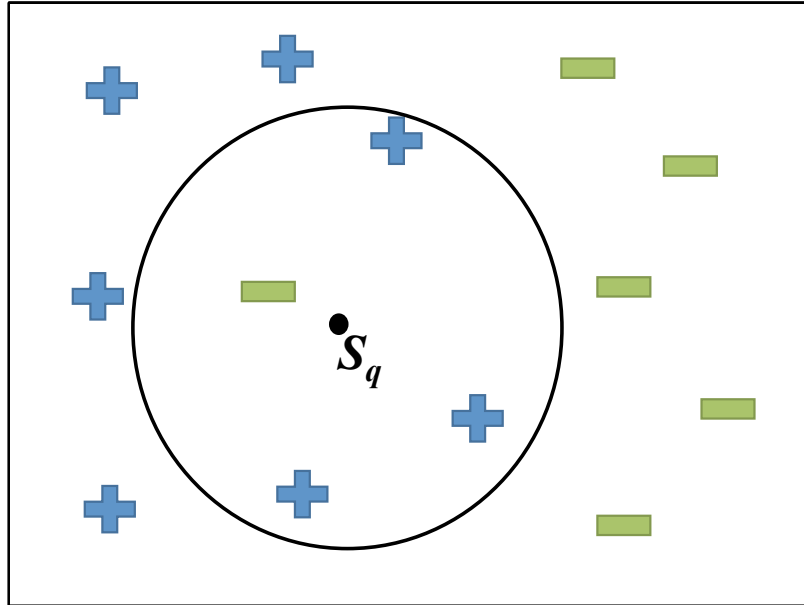


Figure 3.4: The 4-Nearest Neighbor classifier labels a query sample according to the majority vote among its four nearest neighbors in the training database. In this example, three out of four nearest neighbors have positive labels, therefore, S_q is classified as positive.

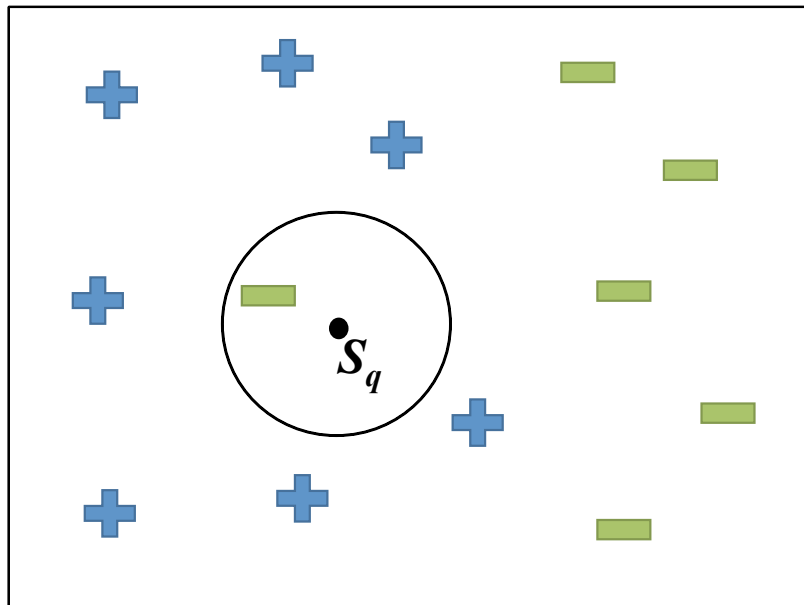


Figure 3.5: The 1-Nearest Neighbor classifier assigns to a query sample the label of its nearest neighbor among training samples. In this example, the nearest neighbor to the query sample has a negative label, therefore, S_q is classified as negative.

3.4.2 Semi-classification-based approach

In the design of our two-step algorithm, we use the term *K nearest neighbors* in a slightly different sense. While the core concept of prioritizing the training instances based on their distance from the query sample remains the same as in the original algorithm, there is no voting process involved, even in the case where K is more than one. Instead, the K closest points to the query sample are stored in a list and are treated as K candidate classes. To distinguish between these two concepts, hereafter we denote the number of candidates by \tilde{K} instead of K .

The main reason for this modification is that sometimes only one training sample from each class is present in the database, hence with the conventional method we would have to classify the testing sample under the closest label and reject all the other labels. In cases where there is some degree of overlap between attributes of different training instances, this method could give rise to significant classification error. In such cases, the distance between instances, while still being a meaningful classification criterion, is no longer the only factor that differentiates between classes. The classification error can be reduced by considering more than one neighboring point, not as target classes, but as a ranked list of class candidates. Indeed, the generated list has to undergo further processing if the target class is to be determined. This post-processing stage forms the subject of Section 3.5.

3.4.3 Alternative distance measures

While the most common, Euclidean (2-norm-based) distance is not the only distance measure that has been used in the K-Nearest Neighbor classification method. A variety of distance measures including the city block (1-norm-based) distance and the correlation distance have also been suggested in the literature, based on specific properties of feature vectors and relative spacial distribution of classes.

The city block distance is defined as the 1-norm of the difference between two feature vectors:

$$d_{CB}(S^{(i)}, S^{(j)}) \equiv \|S^{(i)} - S^{(j)}\|_1, \quad (3.9)$$

which can be computed from

$$d_{CB}(S^{(i)}, S^{(j)}) = \sum_{h=1}^n |S^{(i)}(h) - S^{(j)}(h)|. \quad (3.10)$$

The fundamental difference between the Euclidean and city block distance measures is in the way they partition the space. With the Euclidean distance measure, training samples that are equidistant from a query sample are located on a circle centered around the query sample, whereas with the city block distance measure, instead of a circle, equidistant training samples are located on a square diamond. Unit circles associated with the two distance measures in the two-dimensional Euclidean space are illustrated in Figures 3.6 and 3.7.

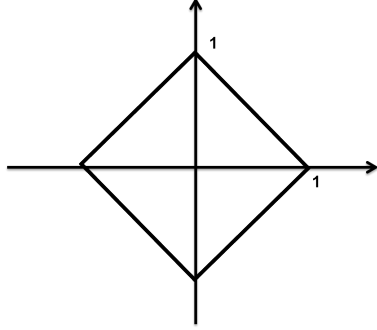


Figure 3.6: Illustration of the unit circle in 1-norm (city block distance measure).

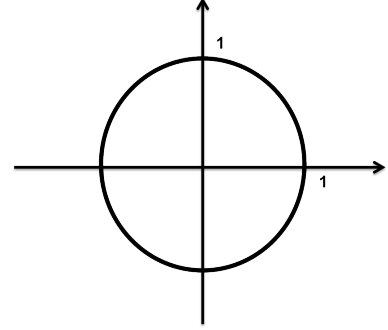


Figure 3.7: Illustration of the unit circle in 2-norm (Euclidean distance measure).

Since the feature vectors in our problem are basically composed of spectral components of musical signals, the use of correlation distance offers quite intuitive interpretation as well as interesting results. The correlation distance between two samples is defined as

$$d_{Corr}(S^{(i)}, S^{(j)}) \equiv 1 - \langle S^{(i)}, S^{(j)} \rangle, \quad (3.11)$$

where $\langle \cdot \rangle$ indicates the inner product operator between two feature vectors

$$\langle S^{(i)}, S^{(j)} \rangle = (S^{(i)})^T S^{(j)}. \quad (3.12)$$

It is important to note that feature vectors in (3.11) are normalized such that their inner product takes on a value in the interval $[0, 1]$. Consequently, the farthest distance value between two samples would be equal to one, which corresponds to the case where the feature vectors are orthogonal.

The correlation distance measure has a three-fold interpretation when applied to spectral feature vectors: (1) Geometric point of view - interprets the distance as the length of orthogonal projection of the testing feature vector on the training feature vector. The best match (zero distance) happens when the vectors are in the same direction, that is when the angle between them is equal to zero. (2) Statistical point of view - interprets the distance as the correlation between random vectors at zero lag. The closer the values of the corresponding attributes, the more alike the spectral patterns, and hence the lower the distance value. (3) Signal processing point of view - interprets the distance as the output of matched filters. Each of the training feature vectors is considered as a matched filter which passes the frequency components related to a certain pitch and blocks the rest of the components. If the spectral peaks in a testing feature vector are located at the same frequencies as those of a training feature vector, the matched filter outputs a value close to one, pointing to a high level of similarity between the received signal and the replica, or equivalently a short distance between samples.

3.4.4 Spectrum normalization

Training and testing samples are usually recorded with different sound levels, which results in scaling issues in distance computation. To solve this problem, we use some form of feature vector normalization prior to computation of each distance measure.

In calculation of the Euclidean and city block distance measures given by (3.7) and

(3.10) respectively, minimum distance value is anticipated when spectral peaks at the same frequencies cancel each other at the subtraction stage. If both training and testing feature vectors are normalized to their maximum value, which is usually assumed to be the power content at the fundamental frequency, then only the place of harmonics on the frequency axis and their relative strengths contribute to the value of the distance between two samples, and the effect of the scaling factor would be minimized. The normalized feature vectors with Euclidean distance measure, S_N^{Euc} , and city block distance measure, S_N^{CB} , are given by

$$S_N^{Euc} = \frac{S}{\|S\|_\infty}, \quad (3.13)$$

$$S_N^{CB} = \frac{S}{\|S\|_\infty}, \quad (3.14)$$

where $\|\cdot\|_\infty$ returns the element with maximum absolute value in a vector.

As explained in Section 3.4.3, the inner product of feature vectors in (3.12) is assumed to yield a maximum value of one. The normalization method for the correlation distance measure can be more conveniently set up from an alternative definition of inner product given below:

$$\langle S^{(i)}, S^{(j)} \rangle = \|S^{(i)}\|_2 \|S^{(j)}\|_2 \cos(\theta), \quad (3.15)$$

where θ indicates the angle between feature vectors. (3.15) can be rearranged as

$$\left\langle \frac{S^{(i)}}{\|S^{(i)}\|_2}, \frac{S^{(j)}}{\|S^{(j)}\|_2} \right\rangle = \cos(\theta). \quad (3.16)$$

As it can be seen, when the feature vectors are normalized to their 2-norms, the value of the inner product will be equal to the cosine of the angle between them, whose maximum value is equal to one. Moreover, all the feature vectors contain only nonnegative attributes. When divided by the 2-norm, a positive value, the sign of the attributes remains unchanged,

which guarantees the minimum value of the inner product being zero. Therefore, the normalized feature vector for correlation distance measure, S_N^{Corr} , is obtained from

$$S_N^{Corr} = \frac{S}{\|S\|_2}. \quad (3.17)$$

3.4.5 Training database

As explained in Section 3.3.3, feature vectors used in the pitch classification stage are essentially certain columns of the spectrogram. Our training dataset is composed of two separate subsets, both containing recorded tones from every piano key (88 recorded tones per piano). The first subset includes tones from only one piano (Steinway grand piano - 88 samples in total). The second subset includes tones from ten different pianos (eight Steinway grand and two Steinway upright pianos - 880 samples in total). All of the recordings are sampled at 44.1 kHz. Training feature vectors are generated by computing the spectrogram of each recorded tone, extracting several columns from the sustain part of the note event and averaging over columns. Note classes are labeled by MIDI numbers, thus the recordings from one piano represent 88 different classes with labels ranging from 21 to 108. As an example, the recorded piano tone and the feature vector corresponding to the pitch class 69 (A4 with fundamental frequency equal to 440 Hz) are depicted in Figures 3.8 and 3.9 respectively.

3.4.6 Testing database

Our testing database includes 75 monophonic themes (either right or left hand) from J. S. Bach's Inventions (number 1, 2, 4, and 8). The length of the recorded themes are adjusted such that they contain 22 note events on average. Each note event is represented by a testing feature vector produced in the same way as the training feature vectors (see Section 3.4.5).

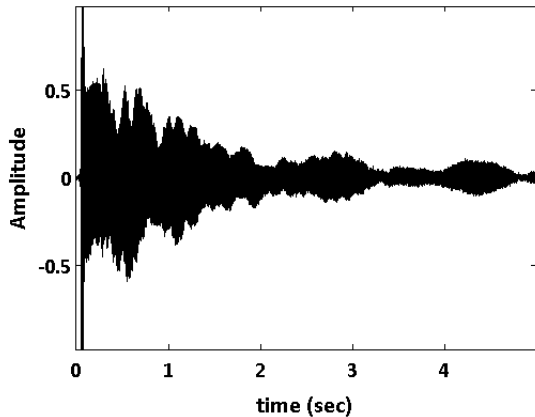


Figure 3.8: Recorded piano tone A4 (with fundamental frequency equal to 440 Hz) in time domain.

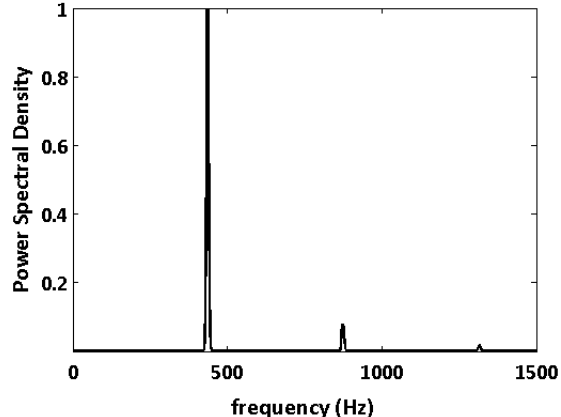


Figure 3.9: Training feature vector from pitch class 69 (normalized to the maximum value).

It should be kept in mind that both training and testing feature vectors contain estimated spectra of musical tones. As a matter of fact, high degrees of overlap among feature vectors' attributes are observed due to harmonicity inherent to this group of signals. For instance, consider two notes that are one octave apart, in which case the spectral components of the higher note are nothing but the even spectral components of the lower one. An illustrative example of this case can be found in Figures 3.10 and 3.11. This characteristic of the musical data justifies the modified approach we have taken towards the original KNN algorithm in Section 3.4.2. In short, the pitch classification task comprises two main steps: first, calculation of the distance between a given testing feature vector and all the training instances retrieved from the memory (*global method*), and second, producing a list of \tilde{K} pitch class candidates in terms of MIDI labels.

3.5 Note sequence detection

In the pitch classification stage, pitch candidates were identified based on the assumption that note events are independent. However, this assumption is not valid in reality. In fact, there exists a strong correlation, governed by musicological rules, between successive notes

in a melodic structure. This can be regarded as another argument in favor of the modified approach toward the KNN algorithm taken in this work. By considering more than one pitch class at each time step (for each note event) rather than deciding upon one target pitch class, ($\tilde{K} > 1$), we provide the algorithm with the capability to explore the sequential relationship between note pitches, as well.

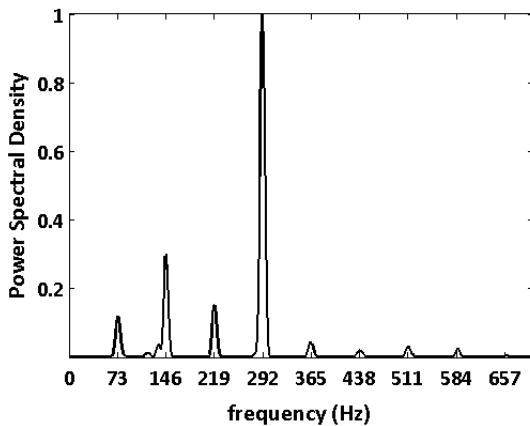


Figure 3.10: Spectrum of the note D2 (MIDI number 18), $F_0 = 73.4$ Hz.

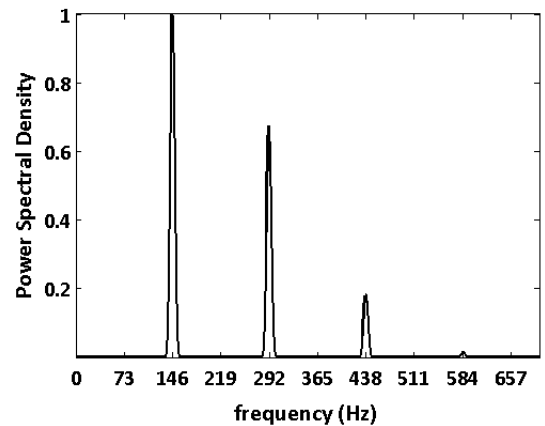


Figure 3.11: Spectrum of the note D3 (MIDI number 30), $F_0 = 146.8$ Hz.

In order to find the melody line composed of pitch candidates in a maximum likelihood scenario, two different sequence estimation methods are considered in our algorithm. The Viterbi algorithm and stack-based tree search are introduced in Sections 3.5.1 and 3.5.2 respectively. In Section 3.5.3, we derive a path metric, a likelihood measure that can be used in either sequence tracking algorithms. We also propose some method-specific modifications to the original version of the path metric to improve sequence tracking results.

3.5.1 Viterbi algorithm

The Viterbi algorithm (VA), first proposed in 1967 for decoding convolutional codes [29], is a recursive method for estimating the state of a stochastic process. The underlying process is assumed to be finite-state, first-order Markov, i.e., the state s_l at discrete time l belongs to a finite collection of \tilde{K} states; in other words, the state space is the set $\{s_1, s_2, \dots, s_{\tilde{K}}\}$. Furthermore, due to the rank of the Markov process, the probability of being in a certain state at time l given all the previous states depends only on the state at time $l - 1$,

$$P(s_l | s_1 \dots s_{l-1}) = P(s_l | s_{l-1}). \quad (3.18)$$

The problem we are trying to solve via the Viterbi algorithm can be stated as finding a sequence of states, \mathbf{s} , that maximizes the *a posteriori* probability (likelihood measure) $P(\mathbf{s} | \mathbf{y})$, or equivalently maximizes $P(\mathbf{s}, \mathbf{y}) = P(\mathbf{s} | \mathbf{y})P(\mathbf{y})$, where \mathbf{y} is a series of observations of a discrete-time, finite-state Markov process. An alternative interpretation of this procedure is finding the shortest path on a *trellis* structure whose nodes correspond to distinct states at given time steps. In this context, the sequence *length*, λ , should obviously have an inverse relationship to the likelihood measure. It can be defined as a negative log likelihood:

$$\lambda = -\ln P(\mathbf{s}, \mathbf{y}) = -\ln P(\mathbf{s} | \mathbf{y}) - \ln P(\mathbf{y}) \quad (3.19)$$

Figure 3.12 depicts a simple example of a trellis structure for a state sequence of length four, with two distinct states at each time step, along with the output of the Viterbi algorithm, i.e., the shortest path. The Viterbi algorithm is well known and commonly used in the signal processing literature, thus we will leave out the implementation details from this thesis and refer the interested reader to the tutorial paper by G. D. Forney [30] for a comprehensive discussion.

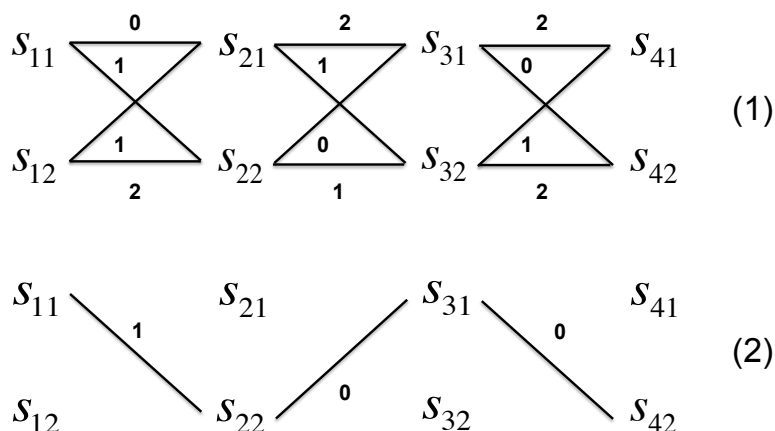


Figure 3.12: (1) Trellis structure for a state sequence of length four, and two states at each time step. $s_{\tilde{k}l}$ denotes the \tilde{k}^{th} state at time step l , and the numbers next to arrows are transition lengths. (2) Output of the Viterbi algorithm - the sequence of states with minimum length.

3.5.2 Stack-based tree search

The stack-based algorithm for tree search can be employed as an alternative to the Viterbi algorithm and often provides complexity reduction when the trellis becomes large. This sequential trial-and error search method, first proposed in the coding literature [31], differs from the basic tree search and the Viterbi algorithm in that instead of the whole tree (or trellis) structure, it only navigates through the most promising paths [32]. A likelihood measure (metric) based on *a priori* information about certain characteristics of the sequence and observations is assigned to every tree branch. In each iteration, only the path corresponding to the largest likelihood value is extended. Since the algorithm deals with a subset of all possible sequences, it would be necessary to keep a list (stack) of extended branches along with their metrics and update it iteratively. In the updated list, the most likely branch in the preceding step is replaced by its children paths.

Let us explain the method with a simple example. The extended tree for a 3-step sequence, starting from a single initial node and exploring only 2 states at each step, is demonstrated in Figure. 3.13 and its corresponding stack in Table. 3.3. By $s_{i,j}$ we denote

the j^{th} state at the i^{th} step, and $m_{i,i'}^{j,j'}$ indicates the metric assigned to the transition from the situation (i, j) , (j^{th} state at the i^{th} step), to (i', j') .

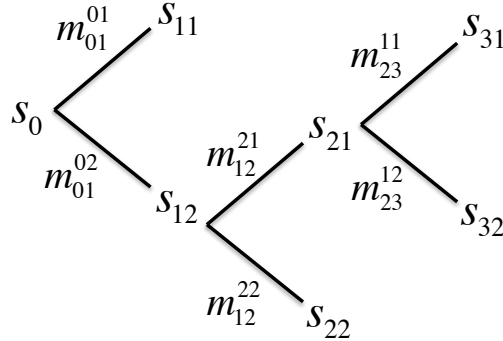


Figure 3.13: Progress of stack-based tree search in a simple case.

Table 3.3: Updated stacks in the example of stack-based tree search.

Stack No.1	Stack No.2	Stack No.3
$s_0 s_{11}$ m_{01}^{01}	$s_0 s_{11}$ m_{01}^{01}	$s_0 s_{11}$ m_{01}^{01}
$s_0 s_{12}$ m_{01}^{02}	$s_0 s_{12} s_{21}$ m_{12}^{21}	$s_0 s_{12} s_{21} s_{31}$ m_{23}^{11}
	$s_0 s_{12} s_{22}$ m_{12}^{22}	$s_0 s_{12} s_{21} s_{32}$ m_{23}^{12}
		$s_0 s_{12} s_{22}$ m_{12}^{22}

In the first step, two paths are extended from the single initial state, s_0 , to both states, s_{11} and s_{12} . These paths along with their corresponding metrics m_{01}^{01} and m_{01}^{02} are entered in the stack number one, the first column in Table 3.3. Then a comparison is made between the two metric entries which returns the second metric as the larger value. Therefore, in the second step we will extend the tree from the node s_{12} . In the updated stack the second

entry is replaced by the two grown paths composed of three nodes and their associated metric values. The comparison is now made between three metric values which returns m_{12}^{21} as the largest. Thus, in the third step, two paths are extended from s_{21} and the stack is updated accordingly. One can see that with the stack-based approach in this simple case, instead of eight possible sequences we ended up exploring only four (half of the full tree size). As the sequence becomes longer and the number of states increases, the tree grows exponentially in size, so the dimensionality reduction offered by this approach seems even more crucial.

In our model, there are \tilde{K} states at each step where each state is in fact a pitch candidate obtained from the classification stage. The length of the sequence, or the number of steps, is equal to the number of detected onsets which would often be much longer than the example we provided here. Considering a total of L steps in a given sequence, the number of all possible sequences, N , can be calculated from

$$N = \tilde{K}^L, \tag{3.20}$$

where all the branches are developed independently and no directional preference has been assumed. In order to set up a stack-based search, an appropriate path metric based on characteristics of our database must be derived.

3.5.3 Path metric derivation

The path metric expresses the likelihood of a note sequence of length L given a series of L observations. The observations in our method are simply testing feature vectors (see Section 3.3.3). Let $S_{1:L}$ and $S_{1:L}^{obs}$ denote the note sequence and observations, respectively. The path metric should then be proportional to the *a posteriori* probability $P(S_{1:L}|S_{1:L}^{obs})$, which according to the Bayes rule can be stated as the product of a likelihood measure and

a priori information about the note sequence:

$$P(S_{1:L}|S_{1:L}^{obs}) = \frac{P(S_{1:L}^{obs}|S_{1:L})P(S_{1:L})}{P(S_{1:L}^{obs})}. \quad (3.21)$$

We assume $P(S_{1:L}^{obs})$ to be a uniform distribution over all possible observations. Therefore, it can be treated as a scaling factor and disregarded in computations. The resulting path metric takes the form of

$$\gamma(S_{1:L}) = P(S_{1:L}^{obs}|S_{1:L})P(S_{1:L}). \quad (3.22)$$

We model the first component in (3.22), $P(S_{1:L}^{obs}|S_{1:L})$, based on spectral information, and the second component, $P(S_{1:L})$, using note transition probabilities obtained from musical data. Details of path metric derivation are presented in Sections 3.5.3 a-c.

(a) Spectral information (Gaussian weights)

The first metric component is derived in a maximum likelihood scenario. In this setup, the observed data consists basically of the spectral information in a testing feature vector at time l . It is referred to as the observed spectrum and denoted by S_l^{obs} in this context. If noise is absent in an acoustic environment, only one of the 88 available spectral vectors in the database is expected to match the observation. However, a completely noise-free recorded piece of music does not exist, and a certain amount of spectrum mismatch is always expected. Let us model the deviation of the observed spectrum from the i^{th} pitch candidate spectrum as

$$S_l^{obs} = S^{(i)} + \eta \quad i = 1, 2, \dots, 88, \quad (3.23)$$

where $S^{(i)}$ and η are $n \times 1$ vectors containing the i^{th} pitch candidate spectrum, and zero-mean uncorrelated Gaussian noise values respectively. The noise probability density function is given by

$$f_{\eta}(\eta) = \frac{1}{(2\pi\sigma^2)^{n/2}} e^{-\frac{\eta^T \eta}{2\sigma^2}}. \quad (3.24)$$

With this noise model, the observed feature vector, S_l^{obs} , is assumed to be drawn from a Gaussian distribution with mean equal to the training feature vector, $S^{(i)}$, and the same variance as in (3.24),

$$f_{S_l^{obs}}(S_l^{obs}) = \frac{1}{(2\pi\sigma^2)^{n/2}} e^{-\frac{(S_l^{obs} - S^{(i)})^T (S_l^{obs} - S^{(i)})}{2\sigma^2}}. \quad (3.25)$$

It should be noted that the choice of Gaussian distribution for modeling spectral deviation is made for convenience. Additionally, multivariate Gaussian distribution has a direct relationship with the Euclidean distance which facilitates its use. The use of distributions matching the properties of the city block and correlation distance measures requires further study. The maximum likelihood scenario can now be described as

$$S_{ML} = \underset{i}{\operatorname{argmax}} \mathcal{L}(S^{(i)} | S_l^{obs}), \quad (3.26)$$

where S_{ML} is the training feature vector maximizing the likelihood function \mathcal{L} defined below

$$\mathcal{L}(S^{(i)} | S_l^{obs}) = f_{S_l^{obs} | S^{(i)}}(S_l^{obs} | S^{(i)}). \quad (3.27)$$

As a simple case, three scalar Gaussian distributions with means $S^{(1)}$, $S^{(2)}$, and $S^{(3)}$ along with the observation, S^{obs} , are shown in Figure. 3.14 . Obviously, the distribution that results in the maximum likelihood value given the observation is the second one, in other words $S_{ML} = S^{(2)}$.

It should be noted that although we set up the likelihood problem for all the available feature vectors in the database ($\mathcal{S}^{(i)}$ for $i = 1, \dots, 88$), it will technically apply to only the \tilde{K} pitch candidates returned by the semi-KNN classifier. This can be considered as somewhat pruning the tree and keeping the most promising states at each time step. Moreover, our primary purpose for this setup, as discussed before, would be to derive some metric that distinguishes more likely paths from less likely. For this reason, instead of selecting the maximizing distribution, we assign to each state or pitch candidate, $\mathcal{S}^{(i)}$ ($i = 1, \dots, \tilde{K}$), the value of the i^{th} distribution for the given observation which can be calculated from (3.25). In the simple case displayed in Figure. 3.14, the red numbers indicate the states (pitch candidates), and the assigned metrics are determined by the distribution values shown by the red dots. Then $S^{(2)}$ and $S^{(3)}$ have the largest and the smallest likelihood measures respectively while $S^{(1)}$ is assigned to a metric between these two values.

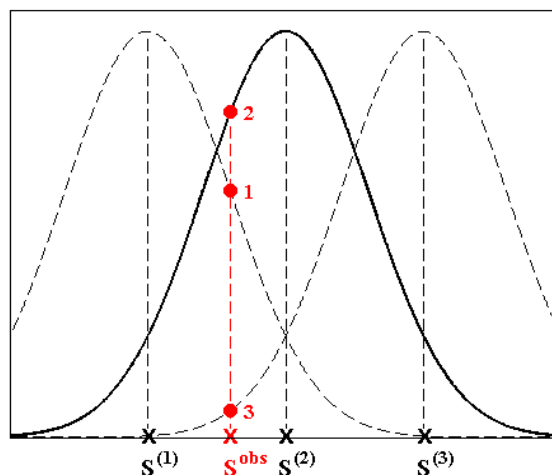


Figure 3.14: Deviation of the observed spectrum from pitch candidate spectra, modeled with Gaussian distributions.

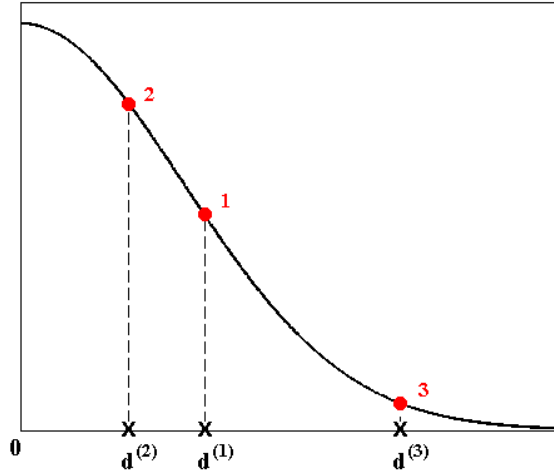


Figure 3.15: Gaussian weights assigned to training feature vectors according to their distance from the observed feature vector.

We could alternatively derive this metric as a refinement to the KNN algorithm. The relationship between (3.8) and (3.25) suggests that the multivariate Gaussian distribution in (3.25) can be mapped into a scalar distribution whose random variable is the Euclidean distance between n -dimensional observed and pitch candidate feature vectors. An equivalent statement is that the candidate feature vectors are given Gaussian weights according to their distance from the observation. Since the Euclidean distance is always a positive value, this new interpretation can be formulated as a half-normal distribution:

$$f_D(d^{(i)}) = \frac{\sqrt{2}}{\sqrt{\pi\sigma_d^2}} e^{-\frac{(d^{(i)})^2}{2\sigma_d^2}}, \quad (3.28)$$

where

$$d^{(i)} = \sqrt{(\mathbf{S}^{obs} - \mathbf{S}^{(i)})^T (\mathbf{S}^{obs} - \mathbf{S}^{(i)})} = \|\mathbf{S}^{obs} - \mathbf{S}^{(i)}\|_2. \quad (3.29)$$

Figure 3.15 illustrates the half-normal distribution associated with the simple example provided in Figure 3.14. The Gaussian weights therefore can be directly obtained from

(3.28) in the classification stage, where the distance values are in any case computed for the purpose of identifying the \tilde{K} nearest neighbors to the observed feature vector.

(b) Note transition probability distribution

In the previous section we developed a metric which assigns Gaussian weights to the \tilde{K} pitch candidates based on the observed spectral information. There is yet another important parameter that must be accounted for in the context of music transcription. As a matter of fact, any melody line can be modeled as a realization of an underlying probability distribution of different note transitions. The probability of a transition from one note to another depends mainly on the distance between the two notes, which in the western music is measured in terms of semitones.

The probability distribution of note transitions has been termed *bigram* in the transcription literature [24]. Prior to its use in music transcription, the term bigram appeared in the context of pattern recognition for text applications [33]. The bigram is typically calculated empirically from a chosen dataset of music samples and provided as *a priori* information. In our case, it is calculated from J. S. Bach’s first three Inventions (see Figure 3.16).

(c) Total metric formulation

The total path metric for a note sequence $S_{1:L}$ of length L is computed according to

$$\gamma(S_{1:L}) = P(S_1^{obs}|S_1) \prod_{l=2}^L P(S_l^{obs}|S_l)P(S_l|S_{l-1}). \quad (3.30)$$

The derived path metric can be employed in both of the note sequence tracking methods explained in Section 3.5. The sequence of pitch classes $S_{1:L}^*$ that maximizes $\gamma(S_{1:L})$, determined via either the Viterbi algorithm or the stack-based tree search, is the resulting

estimate of the note sequence, or melody line.

Unlike the Viterbi algorithm, the stack algorithm does not have a fixed convergence rate. The reason is because when some of the tree branches grow long, their associated path metric values grow very small compared to those of shorter branches. In particular, the algorithm tends to jump back to shorter paths more frequently in the case of very long note sequences, which slows down the convergence rate dramatically.

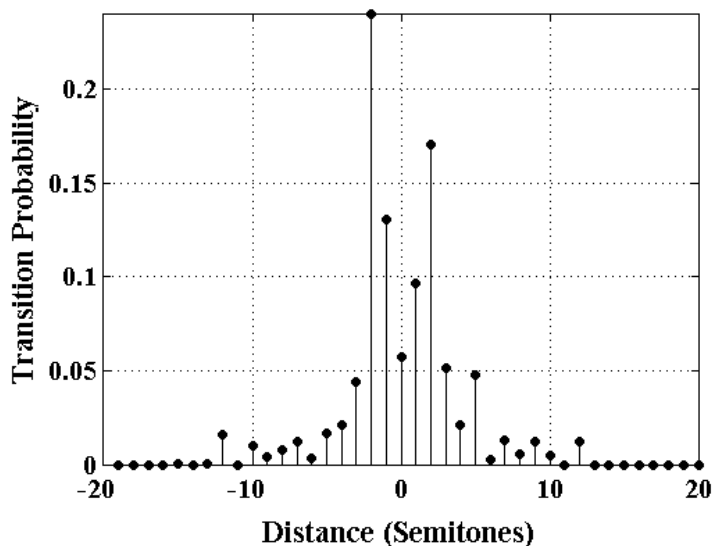


Figure 3.16: Note transition probability distribution (bigram). The values on the x-axis indicate the transition distance in semitones (half notes) and on the y-axis the corresponding probabilities. For example, the probability of moving up one half step between two consecutive notes is roughly 0.1. In our algorithm the bigram is computed empirically from J. S. Bachs Inventions No. 1, 2, and 3.

In order to deal with the convergence issue, the metric values need to be kept in a close range. Thus, we introduce a bias term to the total formulation of the path metric. In general, the bias term attempts to predict the probability of future transitions in each particular direction and combine this information at each step with the current metric values in the stack. In this work, we employ the simplest form of a bias term,

also called a naive bias, which assumes that all of the states (pitch classes) are equally likely to occur at each step in the future.

Let α denote the probability of occurrence of any of the 88 pitch classes ($\alpha = \frac{1}{88}$). The path metric value at step L_1 ($L_1 \leq L$), including the constant bias term can then be stated as

$$\gamma(S_{1:L_1}) = P(S_1^{obs}|S_1) \prod_{l=2}^{L_1} P(S_l^{obs}|S_l)P(S_l|S_{l-1})\alpha^{L-L_1}. \quad (3.31)$$

Chapter 4: Results and Discussion

We evaluate the performance of our proposed method with several system configurations in Section 4.1. The performance accuracies of pitch detection via the conventional KNN algorithm, for different choices of the distance measures and the number of neighboring points, are presented in Section 4.1.1. In Section 4.1.2, we report the performance accuracy of the two step algorithm, composed of a semi-KNN-based pitch candidate classifier and a note sequence tracker. The results are reported separately for the Viterbi and the stack-based tree search algorithms. Finally, we define computational complexity measures for the pitch classification and note sequence tracking algorithms. Computational complexity results are presented, and a comparison between the complexity of proposed algorithms is made in Section 4.2.

4.1 Performance evaluation

4.1.1 Conventional KNN algorithm

The conventional KNN algorithm, which determines the target pitch class for each testing feature vector based on majority vote, is trained over both subsets of the training database (see Section 3.4.5). The first subsets (minimum size) contains only one training sample from each pitch class (88 samples in total), thus with the conventional approach each testing sample is classified according to the label of its closest neighbor in the training dataset (1NN). The second subset contains ten training samples per pitch class (880 samples in total). In this case, more than one neighboring point can be included in the classification process.

The choice of K is somewhat critical to the performance of the KNN algorithm. A small value of K makes the classification results too sensitive to noisy training samples. On the

other hand, a too large value of K undermines the principle based on which the algorithm performs. That is, it blurs the boundary between classes defined in terms of distance. In order to reduce the likelihood of ties in the voting procedure, it would be preferable to consider odd numbers of neighboring points. We set up the algorithm with odd numbers between 1 and 9 (1NN – 9NN). The classification results for both subsets of the training database, and for different numbers of neighboring points included in voting are provided in Table 4.1. We note that the results presented here are obtained using the Euclidean distance measure.

Table 4.1: Note classification performance accuracy for the conventional K-Nearest Neighbor algorithm over two different datasets, and with different number of neighbors included in the voting procedure. Euclidean distance measure is used in all the experiments.

Training dataset	Method (KNN)	Performance accuracy (%)
One sample per class	1NN	75.72
Ten samples per class	1NN	87.52
	3NN	88.87
	5NN	84.57
	7NN	85.92
	9NN	84.69

It can be inferred from the table entries that a larger training database yields a higher performance accuracy in general. This result could be easily predicted, since adding more samples to the training database is equivalent to forming clusters of training points in the Euclidean space, and hence, increasing the likelihood of a given query sample being surrounded by more neighboring points from the target class. Naturally, if the data is not too noisy and the classes are well separated, the vote of the correct target class is more likely to win by including more samples.

In the simplest case, 1NN, the larger dataset results in a roughly 12% performance gain over the smaller dataset. The performance accuracy improves even more (by around 1.3%), if the majority vote is taken among the three closest neighbors. However, extending the proximity area to include 5, 7, or 9 samples has a negative impact on the results. Such behavior could be explained by taking the nature of our data into account. As explained in Section 3.4.6 because of the harmonicity of the musical sound, and hence, the overlapping attributes of the feature vectors, clusters of training points would not be quite well separated. Subsequently, the extended proximity boundary tends to include points from *neighboring clusters*, as well as *neighboring points* form the target cluster, which in part gives rise to misclassification.

Table 4.2: Note classification performance accuracy for the conventional K-Nearest Neighbor algorithm over two different datasets, and with different number of neighbors included in the voting procedure. City block distance measure is used in all the experiments.

Training dataset	Method (KNN)	Performance accuracy (%)
One sample per class	1NN	77.504
Ten samples per class	1NN	91.45
	3NN	88.07
	5NN	85.18
	7NN	86.04
	9NN	85.67

Similar results for the city block (1-norm-based), and correlation distance measures are presented in Tables 4.2 and 4.3. Using the correlation distance measure, the performance accuracy shows similar behavior over increasing values of K to the case where the Euclidean distance measure is used. For $K = 1$, the larger training database results in a roughly 11% performance gain over the smaller training database. The best performance accuracy is observed for $K = 3$, and the accuracy starts decreasing for $K > 3$. The general behavior

of the results with the city block distance measure, however, is different from the previous two cases. The best performance accuracy over the larger training database is achieved for $K = 1$, which is around 14% higher than the performance accuracy of the 1NN method over the smaller database. Furthermore, the city block measure outperforms the Euclidean distance measure by 1.1%, and the correlation distance measure by 0.66% on average.

Table 4.3: Note classification performance accuracy for the conventional K-Nearest Neighbor algorithm over two different datasets, and with different number of neighbors included in the voting procedure. Correlation distance measure is used in all the experiments.

Training dataset	Method (KNN)	Performance accuracy (%)
One sample per class	1NN	76.15
Ten samples per class	1NN	87.40
	3NN	89.36
	5NN	85.12
	7NN	85.92
	9NN	85.98

As discussed in Section 3.4.3, the difference in the results obtained from these three distance measures can be explained considering boundary patterns they create in a high-dimensional space, in other words, the way they separate nearest neighbors to a query sample from the rest of the training samples in the database. In any case, the city block distance measure seems to be the best choice among the three in terms of both performance accuracy and computational complexity, as it yields the best results for the minimum value of K .

4.1.2 Two step algorithm (semi-KNN plus sequence tracking)

In this section we provide the results of our experiments with the two step algorithm, consisting of a semi-classification-based pitch identification stage and a note sequence tracking

stage. We note that all the results of the two-step algorithm are based only on the Euclidean distance measure. Because the bigram in our method is computed from Bach’s Inventions number 1–3 (see Section 3.5.3 (b)), we treat testing samples selected from Inventions number 1 and 2 (40 in total) as the validation set, while the remaining testing samples (35 in total), selected from Inventions 4 and 8, are considered as the true testing set.

Table 4.4: Note classification performance accuracy for the validation and testing sets, both separately and combined. The Viterbi algorithm and stack-based tree search (with and without a naive bias term ($\alpha = \frac{1}{88}$)) are employed at the note sequence tracking stage.

Dataset	Sequence tracking method	Performance accuracy (%) vs. Number of candidates		
		2	3	4
Validation	Viterbi algorithm	93.77	92.76	93.77
	SB tree search (no bias)	93.77	92.76	94.61*
	SB tree search (naive bias)	93.55	92.19	93.43
Testing	Viterbi algorithm	90.71	91.11	90.57
	SB tree search (no bias)	90.71	91.25	91.96*
	SB tree search (naive bias)	88.82	88.96	89.90
Total	Viterbi algorithm	92.37	92.00	92.31
	SB tree search (no bias)	92.37	92.07	93.45*
	SB tree search (naive bias)	91.39	90.71	91.82

Table 4.4 presents the performance accuracy for systems employing either the Viterbi algorithm, or the stack-based tree search at the note sequence tracking stage. The results are reported for the validation and testing sets, both separately and combined. Moreover, the performance of the stack-based tree search is evaluated with and without a naive bias term. The asterisk (*) in the third column of the table indicates removal of some of the sample themes (4 themes out of 75) from the testing database. The sequence tracking

problem seems to be ill-conditioned for these samples, and hence, the hardest to converge. The algorithm was evaluated on the remaining 71 testing samples only. The performance accuracy of the stack algorithm with four candidates, therefore, cannot be compared to that of the other two methods. Nevertheless, the fact that the algorithm is not guaranteed to converge (at least with a reasonable speed) shows the disadvantage of using the stack algorithm without a bias term.

It can be observed from the data in the first column that regardless of the note sequence tracking method used, the two-step algorithm with two candidates yields a total performance accuracy gain of roughly 16% over the 1NN method when given the small training database (See Table 4.1). Increasing the number of candidates seems to have the same impact on both the Validation and the Testing datasets. Adding the third and the fourth candidates does not greatly improve the results, although it does increase the size of the trellis in the Viterbi search or the tree in the stack algorithm, and hence adds to the computational complexity. We will elaborate more on the computational complexity of the proposed algorithms in Section 4.2.

Another interesting observation is the effect of adding a naive bias term to the path metric on the results of the stack-based tree search. The bias term tends to keep the search more focused on longer paths and prevent it from frequently jumping back to shorter ones. As a result, the bias term helps the algorithm converge faster, with a rate comparable to that of the Viterbi algorithm. At the same time, it increases the chance of the search getting stuck in local optima, and hence, deteriorates the total performance accuracy by roughly 1% in each case.

Figures 4.1 and 4.2 present the error rate distribution over piano notes for the two step algorithm with one and two candidates respectively. As can be seen, adding the second candidate not only improves the performance in general, but also alters the behavior of the error distribution, making it more evenly spread. With one candidate, around 42.4% of misclassifications happen above $C4$ (Middle C) and 57.6% below $C4$. With two candidates this portions of the total error change to 46.77% above $C4$ and 53.2% below $C4$. Moreover,

octave error (a note pitch misestimated by 12 semitones) in both cases accounts for over 60% of the total error rate (68.2% in the one candidate case and 64.5% in the two candidate case).

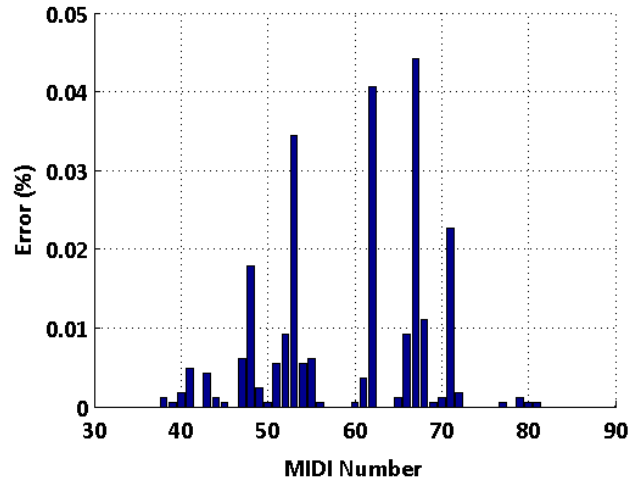


Figure 4.1: Error rate distribution over piano notes for the two-step algorithm with one candidate (technically the 1NN algorithm with the minimum-size database and Euclidean distance measure).

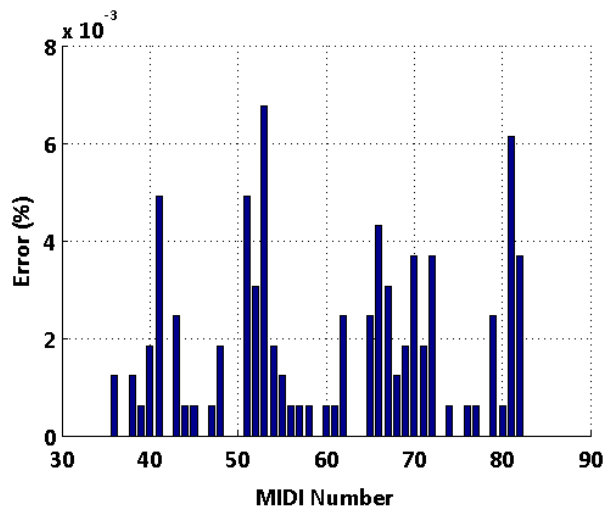


Figure 4.2: Error rate distribution over piano notes for the two-step algorithm with two candidates. Note sequence tracking is carried out via the Viterbi algorithm.

Four illustrative examples are provided in Figures 4.3 - 4.7, demonstrating the impact of different parameter settings on the system output. The effect of the value of K on the classification accuracy is shown in Figure 4.3. The correlation distance measure is used in this example, which confirms the results presented in Table 4.3. With $K = 3$, three out of 26 note pitches are misclassified. Setting $K = 5$ increases the number of misclassifications to five. Interestingly, both of the added misclassifications are octave errors, increasing the total number of octave errors to four. Pitch classes that are one octave apart have common harmonics at even multiples of the fundamental frequency of the lower note, and hence they are very closely spaced. It seems that by extending the search region around the query sample, the immediate neighboring points included would have octave relationship with the target pitch class.

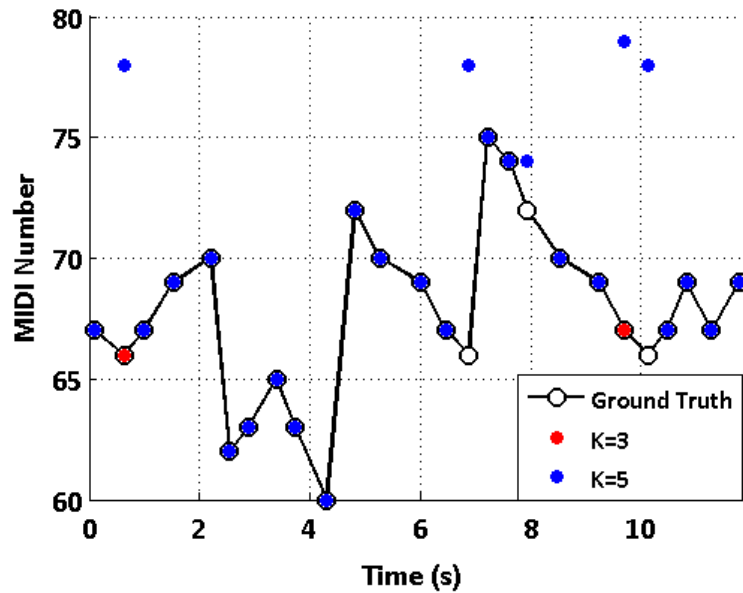


Figure 4.3: Output of the conventional KNN algorithm for a theme of length 26 notes. Correlation distance measure along with the large database (10 samples per class) are employed. With $K = 3$, three notes are misclassified, while setting $K = 5$ increases the number of misclassifications to five.

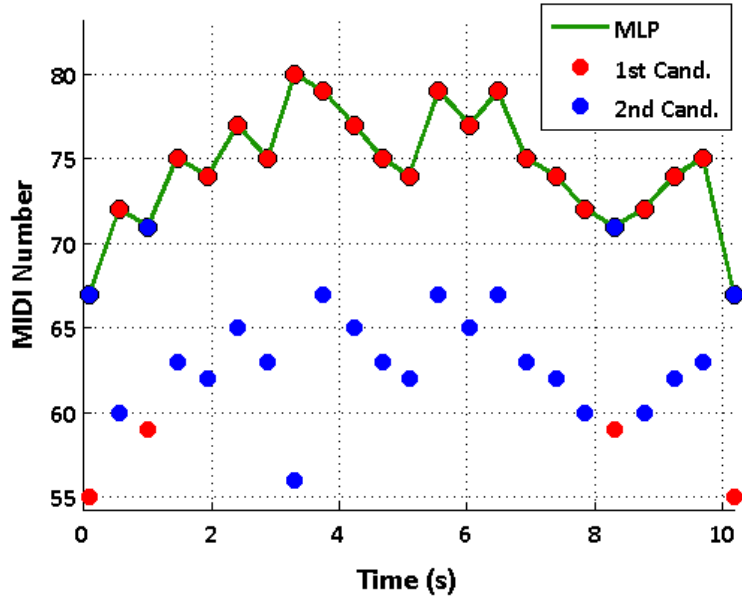


Figure 4.4: Output of the two-step algorithm with one and two candidates for a theme of length 23 notes. In this example adding the second candidate corrects all the three classification errors made using only one candidate.

Figure 4.4 displays the output of the two-step algorithm for a theme of length 23 with one and two candidates. In this example four notes are misclassified when only one candidate is considered. Adding the second candidates, however, results in perfect performance accuracy. Again, all of the four missed note pitches are initially classified under the same note name in one lower octave. For instance, the first note in the sequence is classified as $G3$ (MIDI number 55), while the target pitch class is $G4$ (MIDI number 67). In the case of the example presented in Figure 4.5, adding the second candidate fixes the majority of misclassifications (7 out of 8) but not all of them. In fact, neither of two candidates is the same as the target class in the case of the fourth missed note, thus the note sequence tracker never finds the chance to correct this error.

Figures 4.6 and 4.7 present an example where considering two candidates yields worse performance than only one candidate. With only one candidate 20 out of 23 notes are correctly classified, but adding the second candidate reduces this number to 18.

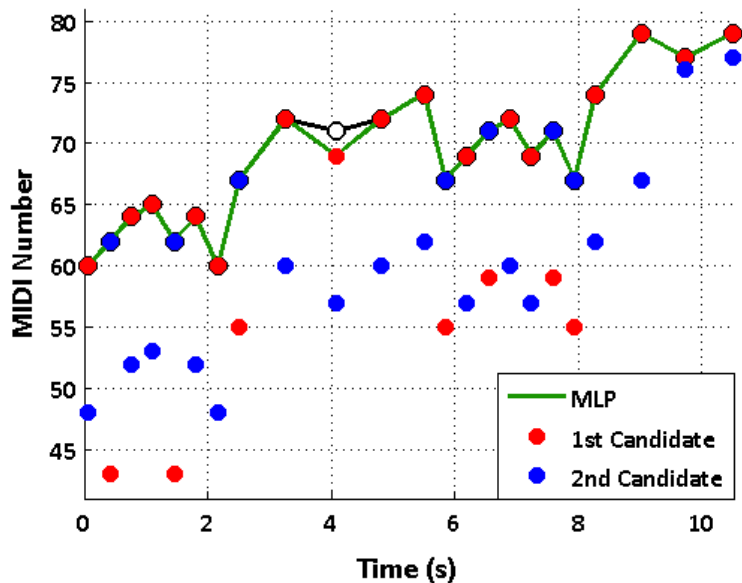


Figure 4.5: Output of the two-step algorithm with one and two candidates for a theme of length 23 notes. In this example adding the second candidate corrects the majority of misclassifications (7 out of 8) but not all of them.

The reason why the note sequence tracker selects the wrong path in between the fourth and the tenth notes is a downward transition of length eleven semitones between notes number 6 and 7 in the actual melody line. The probability of such a transition is zero in the bigram, making the correct path impossible to detect. It should be noted that the melody line in this example is selected from Invention number 4, while the bigram is computed based on the first three Inventions.

Nevertheless, by increasing the number of candidates to three the performance of the algorithm shows an improvement, correctly classifying 21 note events. Although a transition of length -11 is still considered impossible based on the bigram, the third candidate for the note number 6 provides the possibility of making an upward transition of length 8 semitones toward the correct note number seven. Therefore, another impossible transition (upward of length 13 semitones, between the second candidate for note number 6 and the first candidate for note number 7) is replaced by a more likely transition choice (between the

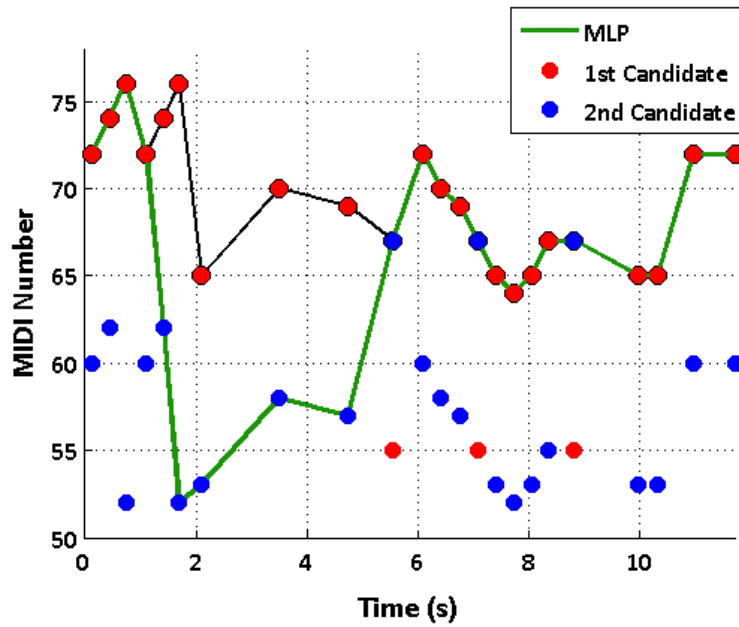


Figure 4.6: Output of the two-step algorithm with one and two candidates for a theme of length 23 notes. Adding the second candidate degrades the performance of the system from three misclassifications to five.

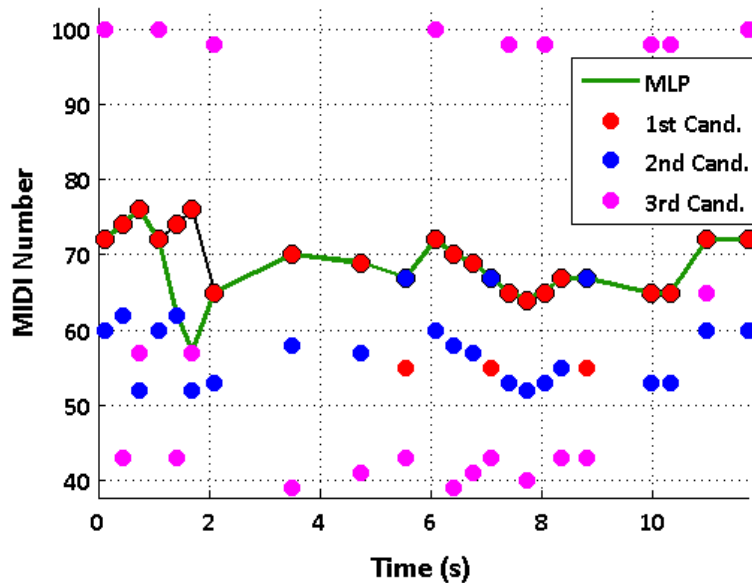


Figure 4.7: Output of the two-step algorithm for the same example as in Figure 4.6. Adding the third candidate fixed three of five misclassifications resulting in 21 correctly classified notes out of 23.

third candidate for note number 6 and the first candidate for the note number 7). This is an interesting example of the case where the note sequence tracker fails to detect correct pitch classes due to limited prior information.

4.2 Computational complexity

4.2.1 K-Nearest Neighbor algorithm

The computational complexity of the KNN algorithm, classifying a single query sample, can be measured in terms of the number of distance calculations, which for the global method presented here is equal to the total number of training samples, D . Thus, total computational complexity for a note sequence of length L is simply LD . For instance, the computational complexity over the minimums-size (1 sample per class) database would be $88L$, while over the largest database we employed (10 samples per class), the complexity increases to $880L$. There is a total number of 1627 note events in the testing database. The total computational complexity over the whole testing set can thus be computed by taking $L = 1627$.

4.2.2 Note sequence tracking

We measure the computational complexity of the note sequence tracking algorithm in terms of the number of state transitions, or equivalently the number of metric calculations. The total number of transitions varies with the type of structure used for implementing the note sequence tracking algorithm. Given the number of observed feature vectors, the number of pitch candidates, and the bigram, the Viterbi algorithm estimates the note sequence from a fixed trellis structure, while the stack-based tree search assumes a pruned tree structure, evolving over iterations. In Sections 4.2.2 (a) and (b), we calculate the computational complexity of both methods for our testing database.

(a) The Viterbi algorithm

The computational complexity of the Viterbi algorithm is governed by the number of pitch candidates and the length of the sequence. The total number of transitions, T , in the trellis structure for a given sequence of length L , and with \tilde{K} pitch candidates assigned to each observation can be stated as

$$T = \tilde{K}^2(L - 1). \tag{4.1}$$

According to (4.1), the number of transitions increases linearly with the length of the note sequence, while this increase is quadratic with the number of pitch candidates. The quadratic growth follows from the fact that \tilde{K}^2 branches are explored in each transition from one note event to the next. We also define the average number of transitions per testing sample as

$$T_{avg} = \frac{\sum_{t=1}^{\mathcal{T}} T_t}{\mathcal{T}}, \tag{4.2}$$

where \mathcal{T} denotes the total number of themes in the database ($\mathcal{T} = 75$ for our database).

(b) Stack-based tree search

The total number of transitions in the full tree structure corresponding to a note sequence of length L with \tilde{K} pitch candidates at each step is given by equation (3.20). It should be obvious that the conventional tree search is computationally more burdensome than the Viterbi algorithm, as the tree size grows exponentially by both the sequence length and the number of pitch candidates. The stack algorithm manages to reduce the computational burden by adopting a trial and error strategy. However, because of uncertainties involved in the process of extending the branches, it is not possible to state the total number of transitions by a closed form equation. Instead, we

calculate the number of extended branches for each testing sample and average over all samples in the testing database.

Table 4.5 presents the average number of transitions over the testing database for different numbers of pitch candidates. While the complexity of all the three algorithms increases by the number of candidates, the complexity of the stack algorithm is much higher than the Viterbi algorithm in each case; it also grows in much faster rate. Including a naive bias term in the stack algorithm reduces the complexity significantly, down to one tenth in the 2-candidate case, one twentieth in the 3-candidate case, and one thirtieth in the 4-candidate case. In spite of a slight reduction in performance accuracy (1%) caused by the bias term, the level of efficiency it introduces to the algorithm justifies its use.

Table 4.5: Computational complexity of the Viterbi algorithm and the stack-based tree search in terms of average number of trellis or tree transitions (or metric computations) per testing sample (theme). Testing samples are 22 notes long on average.

Sequence tracking method	T_{avg} vs. Number of candidates		
	2	3	4
Viterbi algorithm	84	189	336
SB tree search (naive bias)	304	866	5961
SB tree search (no bias)	3190	19606	$> 2 \times 10^5$

4.2.3 The one-step algorithm versus the two-step algorithm

As discussed in Section 4.2.1, we measure the computational complexity of the one-step algorithm (the conventional KNN algorithm) in terms of the total number of distance calculations. For a note sequence of length L and a training database of size D , the total number of distance calculations would be LD . The average length of the themes in our

testing database, L_{avg} , is equal to 22. The average complexity over the large database ($D = 880$) can therefore be computed as $L_{avg}D = 19360$.

In order to have an idea about the complexity of the two-step algorithm, we combine the number of distance calculations in the candidate selection stage and the number of metric calculations in the sequence tracking stage. For the Viterbi algorithm the average complexity can be computed from $L_{avg}D + \tilde{K}^2(L_{avg} - 1)$ and for the stack-based algorithm one can use the entries of Table 4.5. The average complexity of the two-step algorithm over the minimum-size database and for the three sequence tracking methods used are presented in Table 4.6. According to the table both the Viterbi algorithm and the stack-based tree search including a bias term are less complex than the one-step algorithm.

Table 4.6: Computational complexity of the two-step algorithm combining distance calculations in the candidate selection stage and metric calculations in the sequence tracking stage.

Sequence tracking method	Avg. Comp. vs. Number of candidates		
	2	3	4
Viterbi algorithm	2020	2125	2272
SB tree search (naive bias)	2240	2802	7897
SB tree search (no bias)	5126	21542	$> 2 \times 10^5$

Chapter 5: Conclusion

In this work, we presented an instance-based classification approach towards estimating note pitches in monophonic melody lines. The conventional K-Nearest Neighbor algorithm, one of the best known and commonly used instance-based classification methods, was used in order to identify the pitch class of a given note event, based on the distance between the observed frequency-domain feature vector and training feature vectors.

We studied the impact of different distance measures, as well as the value of K , on the performance accuracy. Moreover, it was demonstrated that the performance of the conventional KNN algorithm depends to a great extent on the size of the training database. While in the abundance of training samples the algorithm is capable of yielding high performance accuracies, in cases where the training database is small, due to heightened effect of noisy training data, classification results tend to degrade dramatically.

For scenarios where only a small number of training samples are available, we developed a two stage algorithm in order to improve the pitch classification results. Instead of a target pitch class, the first step of the algorithm outputs a list of pitch class candidates, selected in a semi-KNN-based approach, according to the distance between the observed and training feature vectors. In the second step, a sequence tracking algorithm explores the sequential relationship of the pitch candidates incorporating genre-specific prior information about note transition probabilities.

The two-step algorithm was implemented using two different sequence tracking methods: the Viterbi algorithm, and the stack based tree search. We derived a path metric combining spectral information and note transition probability distribution used in both methods. The Viterbi algorithm takes into account all the possible transitions between pitch candidates, whereas the stack algorithm only extends the most promising path at each step. As such, the Viterbi algorithm has fixed computational complexity and always generates the most

likely note sequence. On the other hand, the stack algorithm is not guaranteed to converge as it tries to maximize the path likelihood on a local basis. Nevertheless, when the stack algorithm does converge, its results match those of the Viterbi algorithm.

According to the experimental results presented in Chapter 4, the two-step algorithm, only tested over the minimum-size training database (1 sample per class), outperforms the conventional KNN algorithm over the largest database we had in our disposition (10 samples per class) in terms of accuracy. Additionally, the average computational complexity of the two-step algorithm, employing the Viterbi algorithm or the stack-based tree search with a bias term in the second step, is still lower than the complexity of the KNN algorithm over a large database. A small training database requires less memory than a large database, which is another advantage of the two-step algorithm over the conventional KNN.

The proposed melody line transcription method in this work was only evaluated on the piano sound and genre-specific melodies. Thus, the first open question in developing this algorithm is how well it would perform on a broader range of instrumental and vocal timbres. Melodic structures from other music genres, and generalization of the bigram to higher-order note dependencies would make another subject of study. Furthermore, the potential of measures other than Euclidean distance to improve the result of the two-step algorithm could be explored in future work.

Bibliography

Bibliography

- [1] Arthur Lenoir, Rémi Landais, and Julien Law-To, “Muma: A scalable music search engine based on content analysis,” in *10th International Workshop on Content-Based Multimedia Indexing (CBMI)*, June 2012.
- [2] Emmanouil Benetos, Anssi Klapuri, and Simon Dixon, “Score informed transcription for automatic piano tutoring,” in *20th European Signal Processing Conference (EUSIPCO 2012)*, August 2012.
- [3] Ye Wang and Bingjun Zhang, “Application-specific music transcription for tutoring,” *IEEE MultiMedia*, vol. 15, no. 3, pp. 70–74, July-September 2008.
- [4] Matti P. Ryyänänen and Anssi Klapuri, “Query by humming of midi and audio using locality sensitive hashing,” in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, April 2008.
- [5] Martin Piszczalski and Bernard A. Galler, “Automatic music transcription,” *Computer Music Journal*, vol. 1, no. 4, November 1977.
- [6] Aaron E. Rosenberg, Garol A. McGonegal, Lawrence R. Rabiner, Michael J. Cheng, “A comparative performance study of several pitch detection algorithms,” *IEEE Transactions on Acoustics Speech and Signal Processing*, vol. 24, no. 5, pp. 399–418, 1976.
- [7] Emilia Gómez, Anssi Klapuri, and Benoît Meudic, “Melody description and extraction in the context of music content processing,” *Journal of New Music Research*, vol. 32, no. 1, pp. 23–40, 2003.
- [8] Myron J. Ross, Harry L. Shaffer, Andrew Cohen, Richard Freudberg, and Harold J Manley, “Average magnitude difference function pitch extractor,” *IEEE Transactions on Acoustics Speech and Signal Processing*, vol. 22, no. 5, October 1974.
- [9] “On the use of autocorrelation analysis for pitch detection,” *IEEE Transactions on Acoustics Speech and Signal Processing*, vol. 25, no. 1, February 1977.
- [10] Giuliano Monti and Mark Sandler, “Monophonic transcription with autocorrelation,” in *COST G-6 Conference on Digital Audio Effects (DAFX-00)*, Dec. 2000.
- [11] Alain de Cheveigné and Hideki Kawahara, “YIN, a fundamental frequency estimator for speech and music,” *The Journal of the Acoustical Society of America*, vol. 111, no. 4, pp. 1917–1930, 2002.

- [12] Martin Piszczalski and Bernard A. Galler, “Predicting musical pitch from component frequency ratios,” *Journal of the Acoustical Society of America*, vol. 66, no. 3, pp. 710–720, 1979.
- [13] Judith C. Brown, “Musical fundamental frequency tracking using a pattern recognition method,” *The Journal of the Acoustical Society of America*, vol. 92, no. 3, pp. 1394–1402, 1992.
- [14] Judith C. Brown and Bin Zhang, “Musical frequency tracking using the methods of conventional and ”narrowed” autocorrelation,” *The Journal of the Acoustical Society of America*, vol. 89, no. 5, pp. 2346–2354, 1991.
- [15] Mürsel Önder, Aydın Akan, and Semih Bingöl, “Pitch detection for monophonic musical notes,” in *Third International Conference on Electrical and Electronic Engineering - ELECO*, December 2003, vol. 1.
- [16] Juan Pablo Bello, Giuliano Monti, and Mark Sandler, “An implementation of automatic transcription of monophonic music with a blackboard system,” in *Irish Signals and Systems Conference (ISSC 2000)*, June 2000.
- [17] Anssi Klapuri, “Qualitative and quantitative aspects in the design of periodicity estimation algorithms,” in *Proceedings of the European Signal Processing Conference*, September 2000.
- [18] Anssi Klapuri and Manuel Davy, *Signal Processing Methods for Music Transcription*, Springer Science+Business Media LLC, 2006.
- [19] Matti Ryyänänen, *Probabilistic modelling of note events in the transcription of monophonic melodies*, Ph.D. thesis, Tampere University of Technology, 2004.
- [20] Matija Marolt, “A connectionist approach to automatic transcription of polyphonic piano music,” *IEEE Trans. on Multimedia*, vol. 6, no. 3, pp. 439–449, June 2004.
- [21] Graham E. Poliner and Daniel P.W. Ellis, “A classification approach to melody transcription,” in *Proceedings of the 6th International Conference of Music Information Retrieval*, University of London, September 2005.
- [22] Graham E. Poliner and Daniel P.W. Ellis, “A discriminative model for polyphonic piano transcription,” *EURASIP Journal on Advances in Signal Processing*, June 2006.
- [23] Simon Rogers and Mark Girolami, *A first course in Machine Learning*, CRC Press, 2012.
- [24] Timo Viitaniemi, Anssi Klapuri, and Antti Eronen, “A probabilistic model for the transcription of single-voice melodies,” in *Proceedings of the Finnish Signal Processing Symposium*, 2003.
- [25] Jim Heckroth, *A Tutorial on MIDI and Wavetable Music Synthesis*, Application Note, CRYSTAL a division of CIRRUS LOGIC, 1998.

- [26] G. Heinzel, A. Rüdiger, and R. Schilling, *Spectrum and spectral density estimation by the Discrete Fourier transform (DFT), including a comprehensive list of window functions and some new flat-top windows*, Max-Planck-Institut für Gravitationsphysik, February 2002.
- [27] P. Welch, “The use of fast fourier transform for the estimation of power spectra: A method based on time averaging over short, modified periodograms,” *IEEE Transactions on Audio and Electroacoustics*, vol. 15, no. 2, pp. 70–73, June 1967.
- [28] Tom M. Mitchell, *MACHINE LEARNING*, The McGraw-Hill Companies, Inc., 1997.
- [29] A. J. Viterbi, “Error bounds for convolutional codes and an asymptotically optimum decoding algorithm,” *IEEE Transactions on Information Theory*, vol. 13, no. 2, pp. 260–269, April 1967.
- [30] G. David Forney, “The Viterbi algorithm,” in *Proceedings of the IEEE*, March 1973, vol. 61.
- [31] F. Jelinek, “Fast sequential decoding algorithm using a stack,” *IBM Journal of Research and Development*, vol. 13, no. 6, pp. 675–685, November 1969.
- [32] Jill K. Nelson and Hossein Roufarshbaf, “A tree search approach to target tracking in clutter,” in *12th International Conference on Information Fusion*, 2009.
- [33] J. Raviv, “Decision making in Markov chains applied to the problem of pattern recognition,” *IEEE Transactions on Information Theory*, vol. 13, no. 4, pp. 536–551, October 1967.

BIOGRAPHY

Fatemeh Pishdadian received her B.Sc. in Electrical Engineering from Ferdowsi University of Mashhad, Mashhad, Iran, in 2005. She has always been in pursuit of an interdisciplinary research area, which combines her two main interests, music and technology. In addition to receiving music theory and piano lessons since adolescence, she spent four active years on her musical training after graduation in order to prepare further for her research field of interest. She has been a PhD student in the Department of Electrical and Computer Engineering at George Mason University since 2010. Her current research is focused on the application of signal processing methods to the analysis of audio/music. She received her M.Sc. in Electrical and Computer Engineering as a secondary degree in 2014.