

國立臺灣大學電機資訊學院電機工程學系

碩士論文 (初稿)

Department of Electrical Engineering

College of Electrical Engineering and Computer Science

National Taiwan University

Master Thesis (DRAFT)



利用結構性支撐向量機的具音樂表現能力之半自動電腦演奏
系統

A Semi-automatic Computer Expressive Music Performance
System Using Structural Support Vector Machine

呂 行

Shing Hermes Lyu

指導教授：鄭士康博士

Advisor: Shyh-Kang Jeng, Ph.D.

中華民國 103 年 6 月

June, 2014



國立臺灣大學
電機工程學系

碩士論文
(初稿)

利用結構性支撐向量機的具音樂表現能力之半
自動電腦演奏系統

呂
行
撰



國立臺灣大學（碩）博士學位論文 口試委員會審定書

論文中文題目
論文英文題目

本論文係呂行君（R01921032）在國立臺灣大學電機工程學研究所完成之碩士學位論文，於民國 103 年○○月○○日承下列考試委員審查通過及口試及格，特此證明

口試委員：

(簽名)

(指導教授)

_____	_____
_____	_____
_____	_____
_____	_____

系主任、所長 _____ (簽名)

(是否須簽章依各院系所規定)



致謝



中文摘要

請打開並編輯[abstractCH.tex](#)

關鍵字：壹、貳、參、肆、伍、陸、柒



Abstract

Open and edit [abstractEN.tex](#)

Key words:A, B, C, D, E, F, G



Table of Contents

口試委員會審定書	i
致謝	ii
中文摘要	iii
Abstract	iv
Table of Contents	v
List of Figures	viii
List of Tables	x
1 Introduction	1
1.1 Motivation	1
1.2 Goal and Contribution	2
1.3 Chapter Organization	2
2 Previous Works	4
2.1 Various Goals and Evaluation	4
2.2 Researches Classified by Methods Used	7
2.3 Additional Specialties	9
3 Proposed Method	11
3.1 Overview	11

3.2	A Brief Introduction to SVM-HMM	13
3.3	Learning Performance Knowledge	18
3.3.1	Training Sample Loader	19
3.3.2	Features Extraction	19
3.3.3	SVM-HMM Learning	20
3.4	Performing Expressively	23
3.4.1	SVM-HMM Generation	23
3.4.2	MIDI Generation	24
3.4.3	Audio Synthesis	24
3.5	Features	25
3.5.1	Score Features	25
3.5.2	Performance Features	27
3.5.3	Normalizing Onset Deviation	29
4	Corpus Preparation	32
4.1	Existing Corpora	32
4.2	Corpus Specification	33
4.3	Implementation	36
4.3.1	Score Preparation	36
4.3.2	MIDI Recording	37
4.3.3	MIDI Cleaning and Phrase Splitting	38
4.4	Results	38
5	Experiments, Results and Discussions	43
5.1	Onset Deviation Normalization	43
5.2	Parameter Selection	44
5.2.1	SVM-related Parameters	44
5.2.2	Quantization Parameter	49
5.3	Human-like Performance	51
6	Conclusions	56

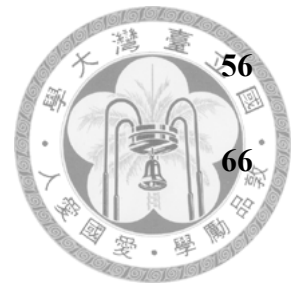


Bibliography

56

A Software Tools Used in This Research

66

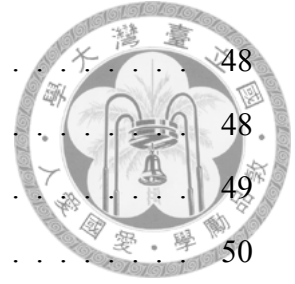




List of Figures

3.1	High Level System Architecture	12
3.2	Hidden Markov Model	17
3.3	Learning Phase Flow Chart	18
3.4	Example input file	22
3.5	Performing Phase Flow Chart	23
3.6	Interval from/to neighbor notes	26
3.7	Relative Duration with the previous/next note	27
3.8	Metric position	27
3.9	Example Score Features	28
3.10	Example Performance Features	29
3.11	Problem with Onset Deviation	29
3.12	The Effect of Automatic Normalization on Onset Deviation feature	31
4.1	Example MusicXML score	35
4.2	Movements Length (in Notes) Distribution	41
4.3	Movements Length (in Phrases) Distribution	41
4.4	Phrase Length (in Notes) Distribution	42
4.5	Example Recording Compared to Score (Pianoroll)	42
5.1	Onset Deviations Using Auto Normalization Method	44
5.2	Onset Deviations By Aligning Last Notes Onset	45
5.3	Onset Deviations By Aligning Last Notes Off	45
5.4	Accuracy for Different ϵ 's	47

5.5	Execution Time for Different ε 's	48
5.6	Accuracy for Different C's	48
5.7	Execution Time for Different C's	49
5.8	Execution Time for different Quantization Granularity	50
5.9	Distribution of Onset Deviation Values from Full Corpus versus Single Performer's Corpus	52
5.10	Distribution of Duration Ratio Values from Full Corpus versus Single Per- former's Corpus	53
5.11	Distribution of MIDI Velocity Values from Full Corpus versus Single Per- former's Corpus	53





List of Tables

4.1	Clementi's Sonatinas Op. 36	34
4.2	Number of Mistakes in Corpus	39
4.3	Total Recorded Phrases and Notes Count	40
4.4	Phrases and Notes Count for Clementi's Sonatina Op. 36	40
5.1	Average Rating for Generated Performance and Human Recording	54
5.2	Average Rating for Generated Performance and Human Recording under Different Part of the corpus	54
5.3	Number of Participants Who Gives Higher Rating to Generated Perfor- mance, Human Recordings or Equal Rating	55
5.4	Number of Participants Who Gives Higher Rating to Generated Perfor- mance, Human Recordings or Equal Rating under Different Part of the Corpus	55



Chapter 1

Introduction

REVIEW1

1.1 Motivation

From the mechanical music performing automata, to the Japanese virtual signer Hatune Miku, there had been many attempts to create automated system that performs music. However, many of these system can only perform predefined expression, which is not very satisfying. State-of-the-art text-to-speech system can already generate fluid and natural speech, but computer performance still can't perform very expressively.

Therefore, many researcher have devoted many effort to develop systems that can automatically or semi-automatically perform music expressively. There is even a bian-annual contest called Music Performance Rendering Contest (RenCon) [1] that puts all performance system to into competition. Their roadmap suggest that they wish to win the Chopin International Piano Contest by a computer performer. We will review previous works, including many which won the RenCon prizes, in Chapter 2.

There are many applications for a computer expressive performance system, many commercial music typesetting software like Finale and Sibelius already have expressive playback features built-in. On the commercial side, such system can provide personalized music listening experience. For the music production industry, it saved a lot of cost on hiring musicians or licensing. Such system can also opens up new opportunity in music

making, such as human-machine co-performance, or interactive multimedia installation. In academia, researchers can use this technology to model the performing style of musicians, or restore historical music archive.



TODO:Discuss Mazolla's theory of expressive performance

1.2 Goal and Contribution

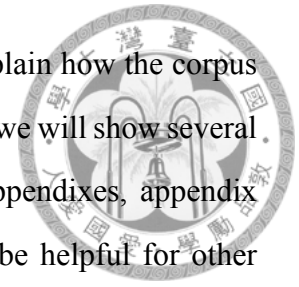
There are many different goals for computer expressive performance, which will be discussed in Chapter 2. In this research the ultimate goal is to be able to play any music in any expressive style specified. But due to the technical and time constrain, we need to set a more practical goal for our research. We wish to build a computer expressive performance system based on an off-line supervised learning algorithm. The system will be able to learn any play monophonic musical phrases. The expressiveness will be at phrase level, structural or timbre related expression are not the primary concern. The performance style will be controlled by the learning material, which is traditional music notation, with human-annotated phrasing information. If only recordings from a single performer is given, it should learn the particular style of the performer.

The major contribution of this thesis is that we apply structural support vector machine on expressive performance problem. There exist no previous work that uses the discriminative learning power of support vector machine on computer expressive performance question. We also developed methods and tools to generate corpus for expressive performance system based on learning algorithms.

1.3 Chapter Organization

In Chapter 2, we will give an overview of previous works and their varying goals, the works will be grouped by the way they learn performance knowledge, and finally we will discuss some additional specialities such as special instrument model or special user interaction pattern. In Chapter 3, we will first give a brief introduction to the mathematical background of the learning algorithm, SVM-HMM. And then we will provide a top-down

explanation to the proposed method. Then in Chapter 4, we will explain how the corpus used for training is designed and implemented. Finally, In Chapter 5, we will show several experiment results and discussions. We have also included two appendixes, appendix A presents some software tools used in this research, which may be helpful for other researchers in music and machine learning fields.





Chapter 2

Previous Works

Researches on computer expressive music performance lags computer composition by almost a quarter of a century, starting around the end of the 1980s [2]. Early work includes the KTH system [2] and other rule-based works. Reviewing computer expressive performance systems is a hard task, because the goals each research wants to achieve has very large differences, which makes evaluation and comparison difficult. To make things worse, there is no training and benchmarking corpus available. Although there is a contest called RenCon [1], which is a music contest for expressive performance systems, they only provide very few audio clips and no shared corpus. So if a system has never entered the contest and doesn't make its source code available, there is no fair way to compare them.

In this chapter, we will first discuss the various goals of computer expressive performance systems, and the difficulties for comparing them. Second, various systems will be briefly introduced by their core algorithm or method used. We will follow the categories proposed by [2]. Interested readers should refer to [2] for more detail. Finally, we will highlight some systems featuring special features like special instrument models.

2.1 Various Goals and Evaluation

The general goal of a computer expressive performance system is to generate expressive music, as opposed to the robotic and dull expression of rendered MIDI. But the definition of “expressive” is very vague and ambiguous, so each research will need to define a

more precise and measurable goal. The following are the most popular goals a computer expressive performance system wants to achieve:



1. Perform music notations in a non-robotic way, regardless of the style.
2. Reproduce a human performance or a musician's style.
3. Accompany a human performer.
4. Validate a musicological theory of expressive performance.
5. Directly render computer composed music works.

Some systems try to perform music notations in a non-robotic way in a general sense, without a certain style in mind. These systems has been employed in music typesetting softwares, like Sibelius [3], to play the notation expressively. Most systems will implicitly achieve this goal.

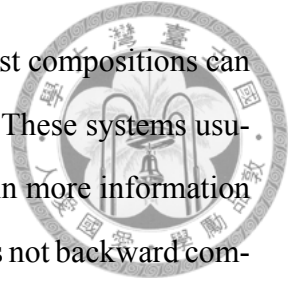
Systems that are designed to reproduce certain human performance or style are usually designed or trained using a particular performer's recording as reference. One commercial example is the Zenph re-performance CD [4], which reproduced the performance style of Rachimaninov, it can perform pieces that Rachimaninov never played in his lifetime in his style.

Accompaniment systems try to render expressive music that act as an accompaniment for a human performer. The challenge is that the system must be able to track the progress of a human performance and render the accompaniment in real-time. One commercial example is Cadenza [?], using the technology created by Christopher Raphel [?]. It claims that it can help music student practice concertos with ease.

Another goal is to validate musicological theories. Musicologist may propose theories on expressive music performance, some of them may want to build a generative model to validate their assumptions. These systems may focus more on the specific phenomenon that the theory tries to explain instead of generating music that is pleasant to human.

Finally, some systems combines computer composition with expressive performance. These systems have a great advantage because the intention of the composition can be

shared with the performance module. Other systems that performs past compositions can only guess the composer's intention by analyzing the score notation. These systems usually has their own data structure to represent music, which can contain more information than traditional music notation, but the resulting performance system is not backward compatible with past compositions.



Because of the high diversity in the goals they want to achieve, the capability of these systems also differs a lot. The capability of a expressive performance system can be broadly categorized into the following three key indicators [2]:

1. Expressive Expression Capability
2. Polyphonic Capability
3. Performance Creativity

Expressive expression capability can range from very high level structural expression (e.g. tempo contrast between sections) to note level expression (e.g. onset, loudness, duration) or even sub-note expression (e.g. loudness envelop, timbre). Most systems can generate note-level expression, but higher or lower level expressions are much rare.

Polyphonic capability indicates if the system can perform polyphonic input. Polyphonic systems are more challenging than monophonic ones because they requires synchronization between voices.

Performance creativity measures the ability of the system to create novel expression. The desired level of creativity varies from goal to goal. A system aiming to recreate human performance may want it to produce fixed expression based on the learning material, while a system that is combined with a composition system may want to create highly novel performance.

REVIEW1 Each system will design different experiment and metrics to verify their goals. Thus, the self-reported results are can't be compared. The only public contest that evaluates expressive performance systems is called RenCon (Performance Rendering Contest) [?]. RenCon is held biannually. Two scores (MIDI) will be given to participants one hour before the competition starts. The participants must generate the expressive

MIDIs in the given time, and then the MIDIs will be played live on a Yamaha Disklavier piano. The audience and a jury consists of professional musicians will give ratings for each performance. The performances are played in random order, so the audience and jury won't know the participant behind each performance, except semi-automatic ones.

The RenCon is divided into fully automatic and semi-automatic categories. But the degree of human intervention in the semi-automatic category varies widely between systems. So it's not very fair to compare them.

TODO: Discuss works that focus on timber only, e.g. Prof. Su's violin work

2.2 Researches Classified by Methods Used

Despite the difference between goals of different expressive performance systems. All of them needs some way to create and apply performance knowledge on inexpressive music. The performance can come from predefined rules or being learned.

Using rules to generate expressive music is the earliest approach tried. Director Musices [5] being one of the early works that is still a living project now. Most of the above systems focus on expressive attributes like note onset, note duration and loudness. But Hermode Tuning System [6] focus more on intonation, thus it can generate expressions that requires string instruments techniques. Pop-E [7] is also a rule-based system which can generate polyphonic music, using its synchronization algorithm to synchronize voices. Computational Music Emotion Rule System [8] put more emphasis on rules that express human emotions. Other systems like Hierarchical Parabola System [5] [9] [10] [11], Composer Pulse System [12, 13], Bach Fugue System [14], Trumpet Synthesis System [15, 16] and Rubato [17, 18] are also some systems that use rules to generate expressive performance. Rule-based systems are effective and don't require a long training period before use. But some of the performance nuance may be hard to describe in rules, so there is a natural limit on how complex the rule-based system can be. Lack of creativity is also a problem for rule-based approach.

Another approach is to acquire performance knowledge by learning. Many machine learning methods have been applied to expressive performance problem. One of the easiest

form is to use linear regression, systems like Music Interpretation System [19--21] and CaRo [22--24] both use linear regression to learn performance knowledge. But assuming the expressive performance as a linear system is clearly not true. So Music Interpretation System use try to solve it by using AND operations on linear regression results to achieve non-linearity. But still linear regression is too simple for this highly non-linear problem.

Many other learning algorithms have been tested with success: ANN Piano [25] and Emotional flute [26] uses artificial neural network. ESP Piano [27] and Music Plus One [28--30] uses Statistical Graphical Models, although the later one focus more on accompaniment task rather than rendering notation. KCCA Piano System [31] uses kernel regression. And Drumming System [32] tried different mapping models that generates drum patterns.

Evolutionary computation has also been applied, genetic programming is used in Genetic Programming Jazz Sax [33]. Other examples include the Sequential Covering Algorithm Genetic Algorithm [34], Generative Performance Genetic Algorithm [35] and Multi-Agent System with Imitation [36, 37]. Evolutionary computation takes long training time, and the results are unpredictable. But unpredictable also means there are more room for performance creativity, so these system can create unconventional but interesting performances.

TODO: Discuss works that focus on timber only, e.g. Prof. Su's violin work Another approach is to use case-based reasoning to generate performance. SaxEx [38--40] use fuzzy rules based on emotions to generate Jazz saxophone performance. Kagurame [41, 42] style (Baroque, Romantic, Classic etc.) instead of emotion. Ha-Hi-Hun [43] takes a more ambitions approach, it's goal is to generate a piece X in the style of and expressive performance example of another piece Y. Another series of researches done by Widmer et al., called PLCG [44--46] uses data-mining to find rules for expressive performance. It's successor Phrase-decomposition/PLCG [47] added hierarchical phrase structures support to the original PLCG system. And the latest research in the series called DISTALL [48, 49] added hierarchical rules to the original one.

Most of the of the performance systems discussed above takes digitalized traditional

musical notation (MusicXML etc.) or neutral audio as input. They have to figure out the expressive intention of the composer by musical analysis or assigned by the user. But the last category of computer expressive performance we will discuss here has a great advantage over the previous ones, by combining computer composition and performance, the performance part of the system can directly understand the intention of the composition. Ossia [50] and pMIMACS [51] are two examples of this category. This approach provides great possibility for creativity, but they can only play their own composition, which is rather limited.

TODO:Fig.:selected figures from previous works

2.3 Additional Specialties

Most expressive performance systems generates piano performance, because it's relatively easy to collect samples for piano. Even if no instrument is specified, the final performance will often be rendered using piano timbre. Some systems generates music in other instruments, such as saxophone [38--40], trumpet [15, 16], flute [26] and drums [52]. These systems requires additional model for the instruments, as a result, they can produce expressions that requires instrumental skills.

The genre of music that a system plays is also a special feature one might have. For systems that doesn't specify the genre, usually western tonal music will be the genre of choice. Composers like Mozart, Chopin and so on well accepted by the public, their scores and literatures are easily accessible. However, both saxophone-based works choose Jazz music, because saxophone is an iconic instrument in Jazz performance. The Bach Fugue System [14], obviously, focus on fugue works composed by bach.

The ability to perform polyphonic music is also a rare feature of computer expressive performance systems. Performing polyphonic music requires synchronization between voices, while allowing each voice to have their own expression. Pop-E [7] use a synchronization mechanism to achieve polyphonic performance. Bach Fugue System [14] is created using the polyphonic rules in music theory about fugue, so it's inherently able to play polyphonic fugue. KCCA Piano System [31] can generate homophonic music -- an

upper melody with an accompaniment -- which is common in piano music. Music Plus One [28--30] is a little bit different because it's a accompaniment system, it adapts non-expressive orchestral accompaniment track to user's performance. Other systems usually generates monophonic tracks only.





Chapter 3

Proposed Method

REVIEW1

3.1 Overview

The high-level architecture of the purposed system is shown in Fig. 3.1. The system has two phases, the upper half of the figure is the learning phase, while the lower half is the performing phase. In the training phase, score and expressive performance recording pairs are feed in, they will serve as training examples for the machine learning algorithm, namely structural support vector machine with hidden Markov model output (a.k.a SVM-HMM). The learning algorithm will produce a performance knowledge model, which can be used to generate expressive performance hereafter. In the performing phase, a score will be given to the system for expressive performance. The SVM-HMM generation module will use the performance knowledge learned in the previous phase to produce expressive performance. The SVM-HMM output then go through a MIDI generator and MIDI synthesizer to produce audible performance.

The system is not intend to add fixed expression to all pieces. Rather it is intended to perform music according to the style which the user wants. This kind of user interactivity can be achieved in two ways: first, the user can choose the training corpus. A user can select a subset of training sample to produce unique models. For example, if the corpus contains only one performer's recordings, the learned model will very likely have a unique

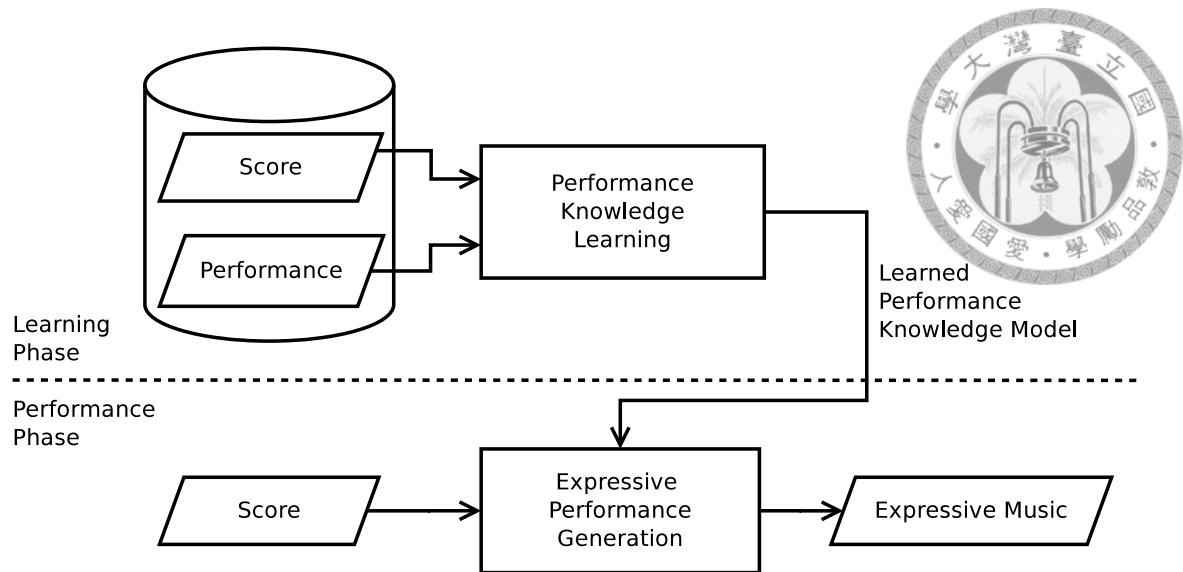


Figure 3.1: High Level System Architecture

style of the performer. Second, the phrasing of a song is given by the user. Since phrasing controls the overall structural interpretation of a music piece, and form a breathing feeling in music, user is given direct control over the performance.

There are still some constrains for the system now. The scores must be monophonic and contains only one musical phrase. One has to manually label the phrasing for any scores used. The learning algorithm, namely SVM-HMM, can only perform off-line learning, so the learning phase can only work in a non-realtime scenario. The generating phase can work much faster though, it can produce expressive music almost instantaneously after loading the score. All the scores are loaded in batch, the system currently don't accept streaming input.

In the following sections, we will discuss the detail steps in the learning and performing phases, and some implementation detail used in each step. Since SVM-HMM learning requires features to be extracted from samples, we will discuss the features used in the end of this chapter.

REVIEW1

3.2 A Brief Introduction to SVM-HMM



In this thesis, we use structural support vector machine (structural SVM) as the learning algorithm. Unlike traditional SVM algorithm, which can only produce univariate prediction, structural SVM can produce structural predictions like tree, sequence and hidden Markov model. Structural SVM with hidden Markov model output (SVM-HMM) is applied to part-of-speech problem with success. This finding lead us to the idea to use SVM-HMM for the expressive performance problem. The part-of-speech tagging problem shares the same concept with expressive performance problem. In part-of-speech tagging, one tries to identify the role by which the word plays in the sentence; while in expressive performance, one tries to determine how a note should be played, according to its role in the musical phrase. To illustrate this, consider a note which is the end of the phrase, which is normally forms a cadence, and a note which is only an embellishment. The first note will probably be played louder and sustain longer than the second note. With this similarity in mind, we believe SVM-HMM will be a good candidate for expressive performance.

The prediction problem in SVM can be described as finding a function

$$h : \mathcal{X} \rightarrow \mathcal{Y}$$

with lowest prediction error. \mathcal{X} is the input features space, and \mathcal{Y} is the prediction space. In traditional SVM, elements in \mathcal{Y} are labels (classification) or real values (regression). But structural SVM extends the framework to generate structural output, such as tree, sequence, or hidden markov model, in this case. To extend SVM to support structured output, the problem is modified to find a discriminant function $f : \mathcal{X} \times \mathcal{Y} \rightarrow \mathcal{R}$, in which the input/output pairs are mapped to a real number score. To predict an output y for an input x , one try to maximize f over all $y \in \mathcal{Y}$.

$$h_{\mathbf{w}}(x) = \arg \max_{y \in \mathcal{Y}} f_{\mathbf{w}}(x, y)$$

Let $f_{\mathbf{w}}$ be a linear function of the form:



$$f_{\mathbf{w}} = \mathbf{w}^T \Psi(x, y)$$

where \mathbf{w} is the parameter vector, and $\Psi(x, y)$ is the kernel function relating input x to output y . Ψ can be defined to accomodate various kind of structures.

In order to apply SVM, we need a way to measure the accuracy of of a prediction, in other words, we have to define a loss function $\Delta : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}$. A loss function has the following property:

$$\Delta(y, y') \geq 0 \text{ for } y \neq y'$$

$$\Delta(y, y) = 0$$

Also, the loss function is assumed to be bounded. Let's assume the input-output pair (x, y) is drawn from a join distrution $P(x, y)$, the prediction problem is to minimize the total loss:

$$R_p^\Delta = \int_{\mathcal{X} \times \mathcal{Y}} \Delta(y, f(x)) dP(x, y)$$

Since we can't directly find the distribution P , we need to replace this total loss with a empirical loss, calculted from the observed training set of (x_i, y_i) pairs.

$$R_s^\Delta(f) = \frac{1}{n} \sum_{i=1}^n \Delta(y_i, f(x_i))$$

With the definition of the loss function ready, we will demonstrate how to extend SVM to structural output, starting with a linear separable, and then extend it to soft-margin formulation.

A linear separable case can be expressed by a set of linear constrains

$$\forall i \in \{1, \dots, n\}, \forall \hat{y}_i \in \mathcal{Y} : \mathbf{w}^T [\Psi(x_i, y_i) - \Psi(x_i, \hat{y}_i)] \leq 0$$

However, in the SVM context, we not only want a solution, but we want the solu-

tion to have the largest margin possible. So the above linear constraints will become this optimization problem



$$\begin{aligned} & \max_{\gamma, \mathbf{w}: \|\mathbf{w}\|=1} \gamma \\ & s.t. \forall i \in \{1, \dots, n\}, \forall \hat{y}_i \in \mathcal{Y} : \mathbf{w}^T [\Psi(x_i, y_i) - \Psi(x_i, \hat{y}_i)] \leq \gamma \end{aligned}$$

, which is equivalent to the convex quadratic programming problem

$$\begin{aligned} & \min_{\mathbf{w}, \xi_i \geq 0} \frac{1}{2} \mathbf{w}^T \mathbf{w} \\ & s.t. \forall i \in \{1, \dots, n\}, \hat{y}_i \in \mathcal{Y} : \mathbf{w}^T [\Psi(x_i, y_i) - \Psi(x_i, \hat{y}_i)] \geq 1 \end{aligned}$$

To address possible non-separable problems, slack variables can be introduced to penalize errors, and result in a soft-margin formalization.

$$\begin{aligned} & \min_{\mathbf{w}, \xi_i \geq 0} \frac{1}{2} \mathbf{w}^T \mathbf{w} + \frac{C}{n} \sum_{i=1}^n \xi_i \\ & s.t. \forall i \in \{1, \dots, n\}, \hat{y}_i \in \mathcal{Y} : \mathbf{w}^T [\Psi(x_i, y_i) - \Psi(x_i, \hat{y}_i)] \geq 1 - \xi_i \end{aligned}$$

C is the parameter for trade-off between low training error and large margin. The optimal C varies between different problems, so experiment should be conducted to find the optimal C for our problem.

Intuitively, a constrain violation with larger loss should be penalize more than the one with smaller loss. So [53] proposed two possible way to take the loss function into account. The first way is to re-scale the slack variable by the inverse of the loss, so a high loss leads to smaller re-scaled slack variable.

$$\begin{aligned} & \min_{\mathbf{w}, \xi_i \geq 0} \frac{1}{2} \mathbf{w}^T \mathbf{w} + \frac{C}{n} \sum_{i=1}^n \xi_i \\ & s.t. \forall i \in \{1, \dots, n\}, \hat{y}_i \in \mathcal{Y} : \mathbf{w}^T [\Psi(x_i, y_i) - \Psi(x_i, \hat{y}_i)] \geq 1 - \frac{\xi_i}{\Delta(y_i, \hat{y}_i)} \end{aligned}$$

The second way is to re-scale the margin, which yields

$$\min_{\mathbf{w}, \xi_i \geq 0} \frac{1}{2} \mathbf{w}^T \mathbf{w} + \frac{C}{n} \sum_{i=1}^n \xi_i$$

$$s.t. \forall i \in \{1, \dots, n\}, \hat{y}_i \in \mathcal{Y} : \mathbf{w}^T [\Psi(x_i, y_i) - \Psi(x_i, \hat{y}_i)] \geq \Delta(y_i, \hat{y}_i) - \xi_i$$



But the above quadratic programming problem has a extreme large number ($O(n|\mathcal{Y}|)$) of constrains , which will take considerable time to solve. So [53] proposed a greedy algorithm to reduce the number of constrains. Initially, the solver starts with an empty working set with no constrains. Than the solver iteratively scans the training set to find the most violated constrains under the current solution. If a constrain violates by more than the desired precision, the constrain is added to the working set. And the solver re-calculate the solution under the new working set. The algorithm will terminate once no more constrain can be added under the desired precision.

In a later work by Joachims et al. [54], they created a new formulation and algorithm to further speed up the algorithm. Instead of using one slack variables each training sample, which results in a total of n slack variables, they use a single slack variable for the n training samples. The follwing formula is the 1-slack version of slack-rescaling structural SVM:

$$\min_{\mathbf{w}, \xi \geq 0} \frac{1}{2} \mathbf{w}^T \mathbf{w} + C\xi$$

$$s.t. \forall i \in \{1, \dots, n\}, \hat{y}_i \in \mathcal{Y} : \mathbf{w}^T [\Psi(x_i, y_i) - \Psi(x_i, \hat{y}_i)] \geq \frac{1}{n} \sum_{i=1}^n 1 - \frac{\xi}{\Delta(y_i, \hat{y}_i)}$$

And margin-rescaling structural SVM:

$$\min_{\mathbf{w}, \xi \geq 0} \frac{1}{2} \mathbf{w}^T \mathbf{w} + C\xi$$

$$s.t. \forall i \in \{1, \dots, n\}, \hat{y}_i \in \mathcal{Y} : \mathbf{w}^T [\Psi(x_i, y_i) - \Psi(x_i, \hat{y}_i)] \geq \frac{1}{n} \sum_{i=1}^n \Delta(y_i, \hat{y}_i) - \xi$$

Detailed proof on how the new formulation is equivalently general as the old one is given in the paper.

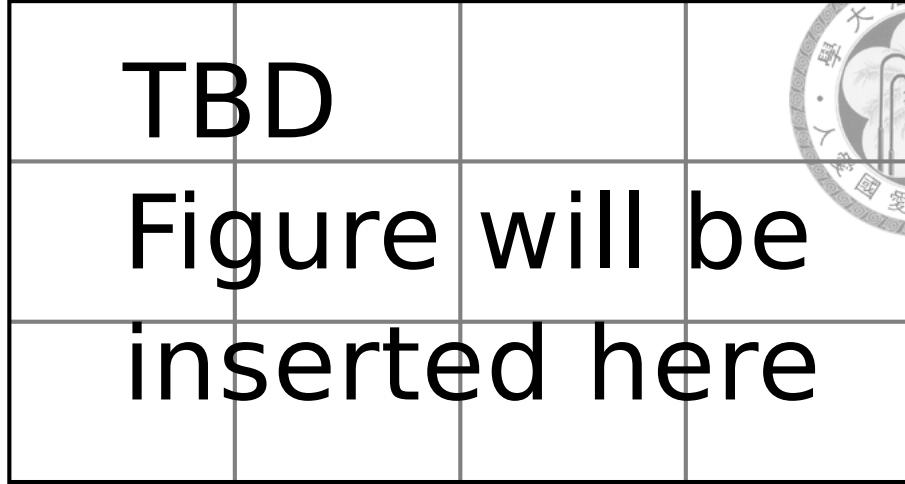


Figure 3.2: Hidden Markov Model

With the framework described above, the only problem left is how to define the general loss function for Hidden Markov Model (HMM)? In [55], the authors proposed two types of features for a equal-length observation/label sequence pair (x, y) . The first is the interaction of a observation with a label, the other is the interaction between neighboring labels.

Formally, for some observed features $\Phi_r(x^s)$ of a note x located in s th position of the phrase, and assume $[[y^t = \tau]]$ denotes the t th note is played at a velocity of τ , the interaction of the two predicate can be written as

$$\phi_{r\sigma}^{st}(\mathbf{x}, \mathbf{y}) = [[y^t = \tau]] \Psi_r(x^s), \quad 1 \leq \gamma \leq d, \quad \tau \in \Sigma$$

And for interaction between labels, the feature can be written as

$$\hat{\phi}_{r\sigma}^{st}(\mathbf{x}, \mathbf{y}) = [[y^s = \sigma \wedge y^t = \tau]], \quad \sigma, \tau \in \Sigma$$

By selecting a dependency order for the HMM model, we can restrict s 's and t 's. For example, for a first-order HMM, $s = t$ for the first feature, and $s = t - 1$ for the second feature. The two features on the same time t is then stacked into a vector $\Psi(x, y; t)$. The feature map for the whole sequence is simply the sum of all the feature vectors

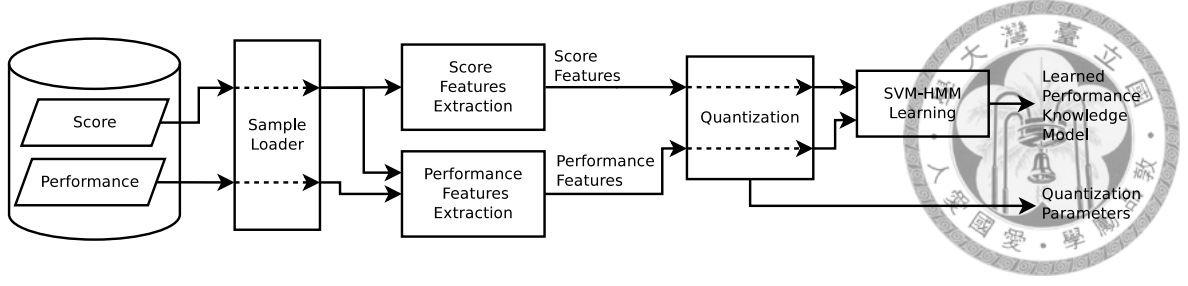


Figure 3.3: Learning Phase Flow Chart

$$\Phi(\mathbf{x}, \mathbf{y}) = \sum_{t=1}^T \Phi(\mathbf{x}, \mathbf{y}; t)$$

Finally, the distance between two feature maps depends on the number of common label segments and the inner product between the input features sequence with common labels.

$$\langle \Phi(\mathbf{x}, \mathbf{y}), \Phi(\hat{\mathbf{x}}, \hat{\mathbf{y}}) \rangle = \sum_{s,t} \left[[y^{s-1} = \hat{y}^{t-1} \wedge y^s = \hat{y}^t] \right] + \sum_{s,t} \left[[y^s = \hat{y}^t] \right] k(x^s, \hat{x}^t)$$

To speed up the computation of F for HMM, a Viterbi-like decoding algorithm is used.

3.3 Learning Performance Knowledge

The main goal in the learning phase is to extract performance knowledge from training samples. Fig. 3.3 shows the internal structure of the learning phase.

Training samples are matched score and expressive performance pairs (their format and preparation process is discussed in Chapter 4). The training samples are loaded as machine-readable format, and have their features extracted. Two types of features will be extracted from the samples, first, the information from the score only are called score features; second, the information of the expressive performance with respect to the score is called the performance features. In order to generate new expressive performance, we want to learn a prediction model by which we can predict the performance features given the score model. This process can be analogize to a human performer reading the explicit and implicit cues from the score, and perform the music with certain expressive expression.

The learning part is done by a supervised learning algorithm. In this thesis, we choose Structural Support Vector Machine with Hidden Markov Model Output (SVM-HMM) as our learning algorithm.



3.3.1 Training Sample Loader

A training sample is loaded with the sample loader module, since a training sample is consisted of a score (musicXML format) and an expressive recording (MIDI format), the sample loader finds the two files given the sample name, and load them into an intermediate representation (`music21.Stream` object provided by the `music21` library [56] from MIT). The `music21` library will convert the musicXML and MIDI format into a python Object hierarchy that is easy to access and manipulate by python code. The loaded score are then handed to the feature extractor module.

One caveat here is that the recording in MIDI may contain very subtle expression. But the `music21` library will quantize the MIDI in the time axis by default, which will destroy the subtle onset and duration expression. And the `music21` library don't handle the “ticks per quarter note” information in the MIDI header [?], so we must explicitly specify that this value, and disable quantization during MIDI loading.

3.3.2 Features Extraction

In order to keep the system architecture simple, a feature extractor are designed to be independent to other feature extractors, so features can added or removed without affecting the rest of the system. And further more, this enables parallel processing during feature extraction. To achieve this, each feature must implement a common feature extractor interface, and feature extractors should not communicate with each other.

But sometimes a feature inevitably depends on other features, for example, the “relative duration with the previous note” is calculated based on the “duration” feature. But since we want to avoid complex scheduling with dependency, the “duration” feature is extracted during the extraction of the “relative duration” feature, no matter if the “duration” feature is extracted prior of after the “relative duration”. Therefore, the “duration” feature

extracted has computed twice. To avoid redundant computation of the feature extractors, we implemented a caching mechanism. Say, the “duration” feature had been computed once, its value will be cached during this execution session; when the “relative duration” extractor calls the “duration” extractor again, the value is retrieved from cache without recalculation. This method can speed up the execution while keeping the system complexity low.

The extracted features are aggregated and stored into a JavaScript Object Notation (JSON) file. By saving the features in a human-readable intermediate file, the researcher can easily look into the file to debug potential problems.

3.3.3 SVM-HMM Learning

After all features are extracted, the next step is to learn performance knowledge from the features. In the early stage of this research, we have tried linear regression with limited success [57]. However, the assumption of linearity is an oversimplification. So we switch to Structural Support Vector Machine with Hidden Markov Model Output (SVM-HMM) [53--55] as our supervised learning algorithm.

The SVM-HMM learning module loads the JSON file from the previous stage, and rearrange the features to fit the required input format of the SVM-HMM learner program. However, the features from the previous stage are real numbers, but SVM-HMM only takes discrete output, so quantization is required here. For each feature, the quantizer calculates the overall mean and standard deviation from all training samples. Quantization has some effect on the final outcome, so it is treated as a way to generate innovative expression. Here we will present a quantizer design for demonstration purpose: We use a uniform quantizer to quantize a performance feature into 128 bins. Then the quantizer divides the range between mean minus or plus four standard deviations into 128 uniform intervals. Values over than mean plus four standard deviations are quantized into the 128th bin, and values below mean minus four standard deviations are quantized into the 1st bin. While reconstruct the features from the quantized value, the middle point of each bin is used. For the 128th bin, the mean plus four standard deviation is used, and similar for the

1st bin. The number of intervals decides how fine the quantization will be, if the number is too low, the quantization error will be large, expressions across a large range will be quantized into the same value, and result in a dull expression. However, if the number is too large, There will be too few samples for each interval, and the training process will take a lot of CPU and memory resources, without significant gain in prediction power. The range of four standard deviation is also subject to adjustment, a narrow range will quantize many values into the largest of smallest bin, so the performance will have a lot of saturated values. But a very large range will make the interval between each quantization bin too large, rising the quantization error.

The theoretical background of SVM-HMM is already described in Section 3.2, to implement the algorithm we leverage Thorsten Joachims's implementation called SVM^{hmm} [58]. SVM^{hmm} is an implementation of structural SVMs for sequence tagging [55] using the training algorithm described in [53] and [54]. The SVM^{hmm} package contains a model training program called `svm_hmm_learn` and a model prediction program called `svm_hmm_classify`, which will be used in the performing phase. For structural simplicity, we train a separate model for each quantized performance feature, each model uses all the quantized score features to try to predict a single performance model. The `svm_hmm_learn` takes a training file describing those features. Each line represents features for a note, organized in the following format:

```
1          PERF qid:EXNUM FEAT1:FEAT1_VAL FEAT2:FEAT2_VAL ... #comment
```

PERF is a quantized performance feature. The EXNUM after `qid:` identifies the phrases, all notes in a phrase will have the same `qid:EXNUM` identifier. Following the identifier are quantized score features, denote as `feature name : feature value`, separated by space. And anything after the `#` is comments. An example of the training file is shown in Fig. 3.4

TODO: partial model

There are some key parameters need to be specified for the training process. First the C parameter in SVM, which controls the trade-off between low training error and large margin. Larger C will result in lower training error, but the margin may be smaller.

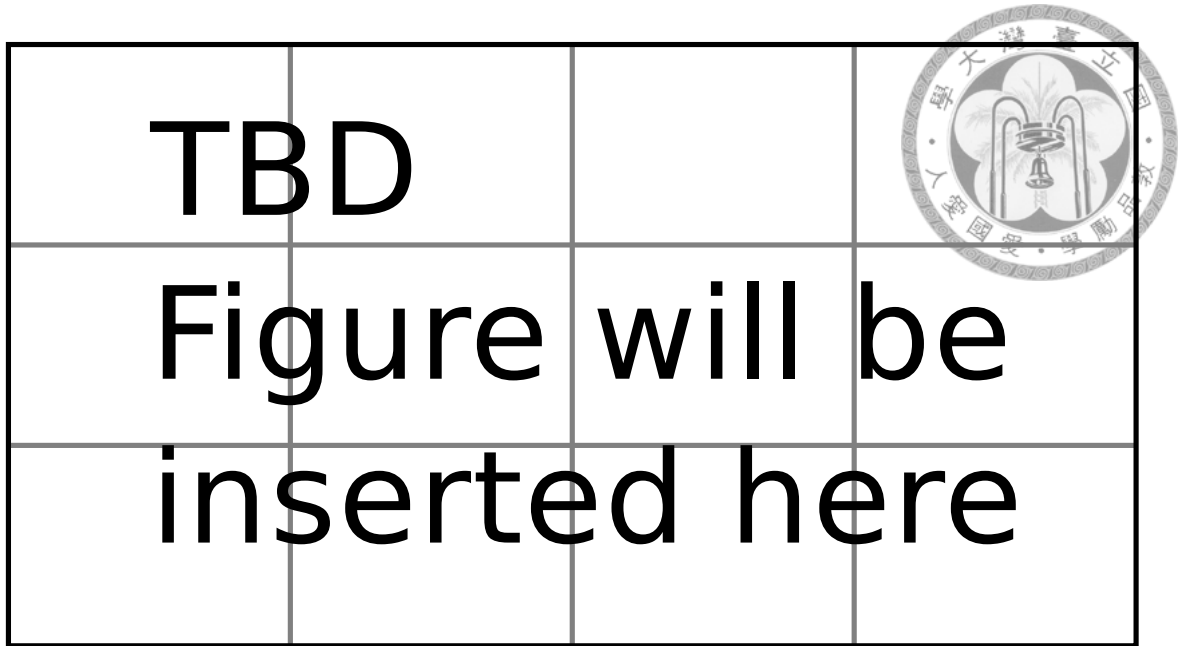


Figure 3.4: Example input file

Second, the ε parameter controls the required precision for the constraints. The smaller the ε , the precision will be higher, but may require more computation. Finally, for the HMM part, we need to specify the order of dependencies of transition states and emission states. In our case, transition dependency is set to one, which stands for first-order Markov property, and emission dependency is set to zero. Since we train separate models for each performance feature, each model can have their own set of parameters. The parameter selection process is done by experiment, which will be presented in Chapter 5

Finally, the training program will output three model files (because we use three performance features). In the model file are binary representation of the SVM-HMM model parameters, such as the support vectors and other informations, which represents the performance knowledge learned. Since it takes considerable time to train a model (depending on the amount of training samples and the power of the computer running the system), the system can only support off-line learning. But the learning process only need to be run once. The performance knowledge model can be reused over and over again in the performing phase.

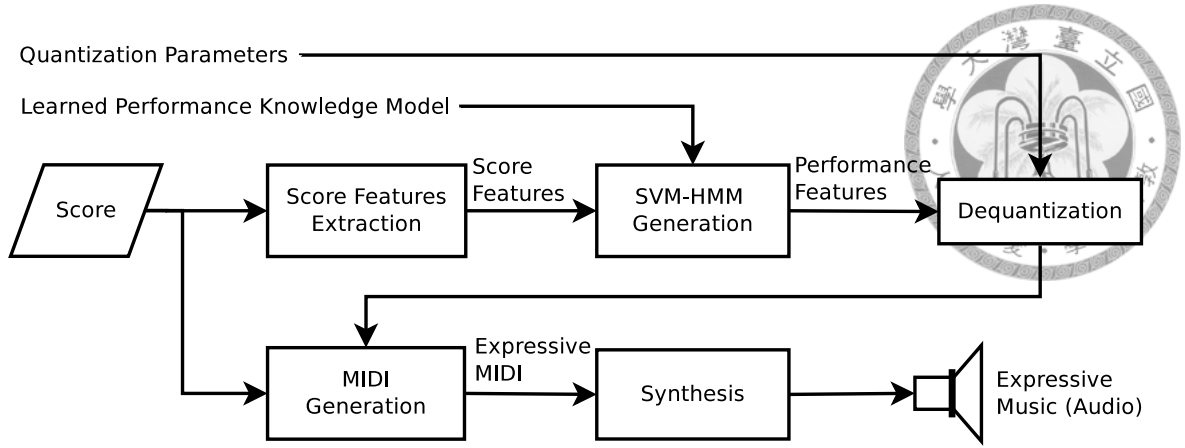


Figure 3.5: Performing Phase Flow Chart

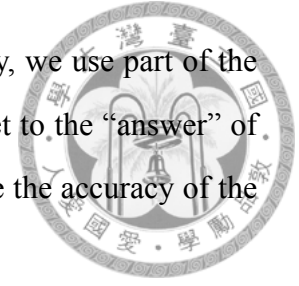
3.4 Performing Expressively

With the performance knowledge learned, we can start to perform music expressively. The performing phase share the same sample loader and feature extractor module with the learning phase. But in the performing phase, the input is only a score file (musicXML containing a single phrase), which is loaded and has its score features extracted. The extracted score features are also stored into a JSON file for the SVM-HMM generation module to load. The performance knowledge model is also an input in this phase. The SVM-HMM generation module will use the learned model and the quantized score features to determine the performance features for the given score. An MIDI generation module will apply those performance features onto the musicXML score to produce a expressive MIDI file. The MIDI file itself is already a expressive performance, in order to listen to the sound, an software synthesizer is used to render the MIDI file into WAV or MP3 format.

3.4.1 SVM-HMM Generation

As in the learning phase, the score features are stored in a JSON file. The SVM-HMM generation module first load this JSON file, and preform the same quantization as the learning phase. The quantized score features are then transformed into the same format as the training file, but the PERF fields are all set to zero, meaning that we don't know its value and wish the algorithm to predict it. The `svm_hmm_classify` program will take these inputs with the learned model file and predict the quantized labels of the performance

features. If we already know the real performance feature value, say, we use part of the corpus as training set and the rest as testing set, the `PERF` can be set to the “answer” of the testing set, so the `svm_hmm_classify` program will calculate the accuracy of the prediction.



3.4.2 MIDI Generation

Since the output of the SVM-HMM learner is the quantized label for the performance feature, dequantization is required to turn those values back to real-valued performance features. The dequantizer will load the quantization parameters from the learning stage to understand the range and intervals used in the quantizer. Each quantization label is dequantized into the mean value of the interval it belongs.

The performance features are then applied onto the input score. For example, if a performance feature represents the note's duration should last for 1.2 times of its nominal value, the duration in the score is multiplied by 1.2. After all the performance features are applied, the expressive version of the score is stored in MIDI format using the `music21` library.

TODO:Dramatization/post processing

3.4.3 Audio Synthesis

In order to actually hear the expressive performance, the MIDI file is then rendered by a MIDI synthesizer. Since the output is standard MIDI file, the user can choose any compatible software or hardware synthesizer. We choose `timidity++` software synthesizer for this job. The output is an WAV(Waveform Audio Format)file, which is then compressed into MP3 (MPEG-2 Audio Layer III) by `lame` audio encoder.

Because sub-note-level expression is not the primary goal of this research, we choose to use standard MIDI synthesizer to render the music. The system can be extended to use more advanced physical model or musical instrument specific audio synthesizer. Sub-note level features, such as special techniques for violins, can be added to the features list and be learned by the SVM-HMM model.

TODO: phrase concatenation



3.5 Features

As mentioned in Section 3.3, there are two types of features, score features and performance features. We will present the features used in the system, and discuss the difficulties encountered.

3.5.1 Score Features

Score features are musicological cues presented in the score. The purpose of score features are to simulate the high level information a performer may perceive when she reads the score. The basic time unit for the features are notes. Each note will have one of each features presented below. Score features includes:

Relative position in a phrase: The relative position of a note in the phrase, its value ranges from 0% to 100%. Since there are often salient musical expressions during the opening or closing of a phrase, this feature is used to capture the start or end of the phrase.

Relative pitch: The pitch of a note relative to the pitch range of the phrase, denoted by MIDI pitch number (resolution is down to semitone). For a phrase of n notes with pitch P_1, P_2, \dots, P_n ,

$$RP = \frac{P_i - \min(P_1, P_2, \dots, P_n)}{\max(P_1, P_2, \dots, P_n) - \min(P_1, P_2, \dots, P_n)}$$

Where P_i is the pitch of note at position t

Interval from the previous note: The interval between the current note and its previous note, in semitone. This represents the direction of the melodic line.

$$IP = P_i - P_{i-1}$$

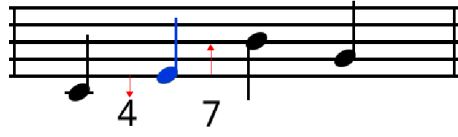


Figure 3.6: Interval from/to neighbor notes

See Fig. 3.6 for example.

Interval to the next note: The interval between the current note and its previous note, in semitone.

$$IN = P_{i+1} - P_i$$

See Fig. 3.6 for example.

Note duration: The duration of a note counted in number of quarter notes.

Although this feature may seem an obvious one, there are still a caveat: Grace notes, which represents ornaments, have zero duration in musicXML specification. The reason for this is that grace notes are considered very short ornaments that does not occupy solid beat position. But zero duration will easily cause divide-by-zero exception in the program, and it's hard to handle in math formulation. So we assigned a duration of 0.0625 quarter note to all grace notes, which is equivalent to the length of a sixty-fourth note. Sixty-fourth note is chosen because it's far shorter than all the notes in our corpus.

Relative Duration with the previous note: The duration of a note divided by the duration of its previous note. For a phrase of n notes with duration D_1, D_2, \dots, D_n ,

$$RDP = \frac{D_i}{D_{i-1}}$$

See Fig. 3.7 for example. This feature is intended to locate local change in tempo, such as a series of rapid consecutive notes followed by a long note.

Relative duration with the next note: The duration of a note divided by duration of its

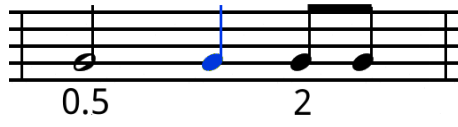


Figure 3.7: Relative Duration with the previous/next note



Figure 3.8: Metric position

next note.

$$RDN = \frac{D_i}{D_{i+1}}$$

See Fig. 3.7 for example.

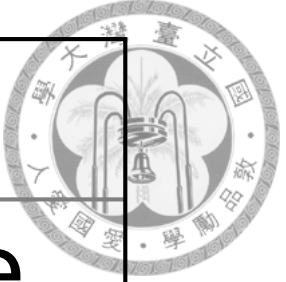
Metric position: The position of a note in a measure, measured by the beat unit defined by the time signature. For example, a $\frac{4}{4}$ time signature will have a beat unit of a quarter note. So if the measure consists of four quarter notes, each of them will have metric position of 1, 2, 3 and 4. Please refer to Fig. 3.8 for example.

Metric position implies beat strength. In most tonal music, there exist a hierarchy of beat strength [?]. For example, in a measure of a $\frac{4}{4}$ piece, the first note is usually the strongest, the third note is the second strongest, and the second and fourth notes are the least strong.

3.5.2 Performance Features

Performance features are the expressive expressions of a performance, which is what we want to learn and generate in this research. Performance features are extracted by comparing the expressive performance with the score. Performance features includes:

Relative onset time deviation: The onset time of a natural human recording will not be exactly as the ones indicated on the score, this phenomenon is roughly corresponding to the music term “rubato”. Given a fixed tempo (beats per second), the score



TBD			
Figure	will	be	
inserted	here		

Figure 3.9: Example Score Features

timing of each note can be calculated as $\text{tempo} \times (\text{beats from the start of phrase})$. The relative onset time bias is the difference of onset timing between the performance and the score, divided by the total length of the phrase. Namely,

$$ROB = \frac{O_i^{perf} - O_i^{score}}{\text{length}(\text{phrase})}$$

Where O_i^{perf} is the onset time of note i in the performance, O_i^{score} is the onset time of note i in the score.

However, the above formula assumes the performance is played at the tempo assigned in the score. In the corpus we use, test subject can't always keep up with the speed of the score because of limited piano skill, or they may speed up or slow down certain sections as their expression. Therefore, the performance should be linearly scaled to avoid such problem, We will discuss this issue in Section 3.5.3.

Relative loudness: The loudness of a note divided by the maximum loudness in the phrase. Measured by MIDI velocity level 0 through 127.

$$RL = \frac{L_i}{\max(L_1, L_2, \dots, L_n)}$$

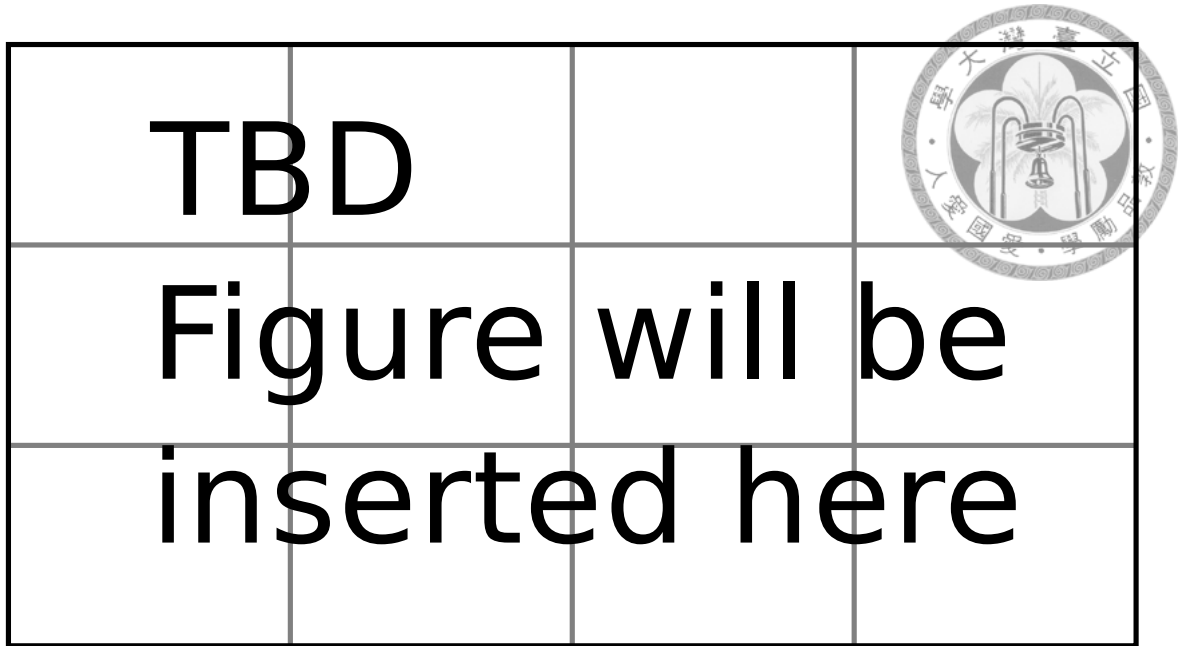


Figure 3.10: Example Performance Features

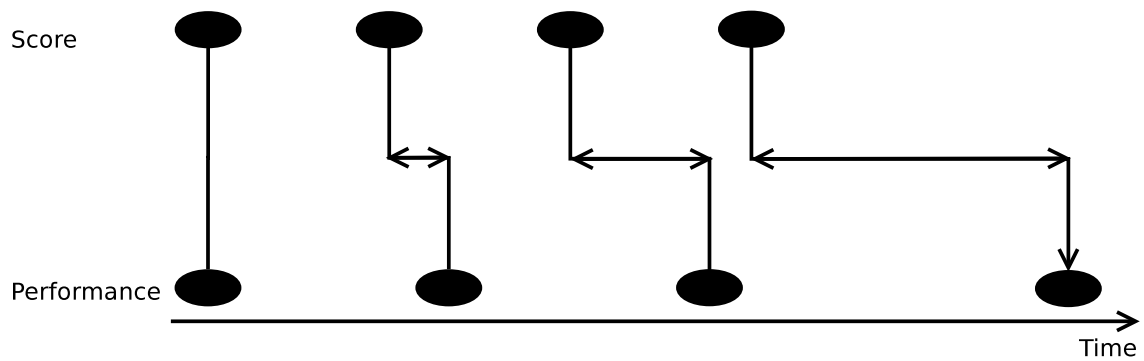


Figure 3.11: Problem with Onset Deviation

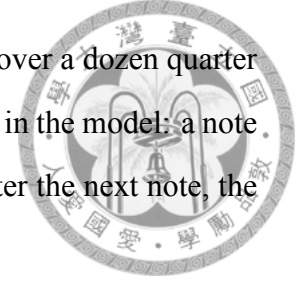
Relative duration: The performed duration of note divided by the duration indicated in the score.

$$RD = \frac{D_i^{perf}}{D_i^{score}}$$

3.5.3 Normalizing Onset Deviation

In previous section, we mentioned that the onset deviation feature has problems when performance is not exactly the same length as the score. Illustrated in Fig. 3.11, if the performance is played faster than expected, the deviation will grow larger and larger over

time. For a long phrase, the onset deviation of the last notes may be over a dozen quarter notes. These kind of extreme values will cause erroneous predictions in the model: a note may be delayed for a very large deviation, causing it to be played after the next note, the swapped notes will destroy the melody.



In other words, the original definition of the onset deviation actually contains two type of deviation: a global deviation cause by difference in the tempo, and a local deviation cause by note-level expression. Since the intention of the onset deviation features is to catch the note-level expression, the performance must be linearly scaled to cancel the global deviation.

Initially, we tried four possible type of normalization methods :

1. Align the onset of the first notes, align the onset of the last notes
2. Align the onset of the first notes, align the end of the last notes

But empirical data shows that nither methods can fit all training samples. One method may produce good result when trained with part of the corpus, but may not be suitable for the rest of the corpus. So we switch to a more dynamic approach: let the program find the best ratio for normalization. To achieve this goal, we first have to define how “fit” two phrases are. If the onset time of all the notes in a phrase are concatenated into a vector, the l^2 -norm of the tow vectors can be treated as the distance. Note that the two vectors must have the same size, because the recordings are required to match note-to-note with the score. So the problem becomes how to find a optimal scaling ratio such that the scaled recording has the minimum distance from the score. The Brent's Method [59] is used to find the optimal ratio. To speed up the optimization and prevent unreasonable value, a search range of $[initial_guess \times 0.5, initial_guess \times 2]$ is imposed on the optimizer. The *initial_guess* is used as a rough estimate of the ratio, calculated by aligning the first and last onset of the phrase. Than we assume the actual ratio will not be smaller than half of *initial_guess* and not larger than twice of *initial_guess*. The two numbers 0.5 and 2 are arbitrary chosen, but most of the empirical data suggest is valid most of the time.

So we have defined a automatic method to dynamically adjust the normalization ratio to eliminate systematical error in the onset deviation feature. Comparing this method to

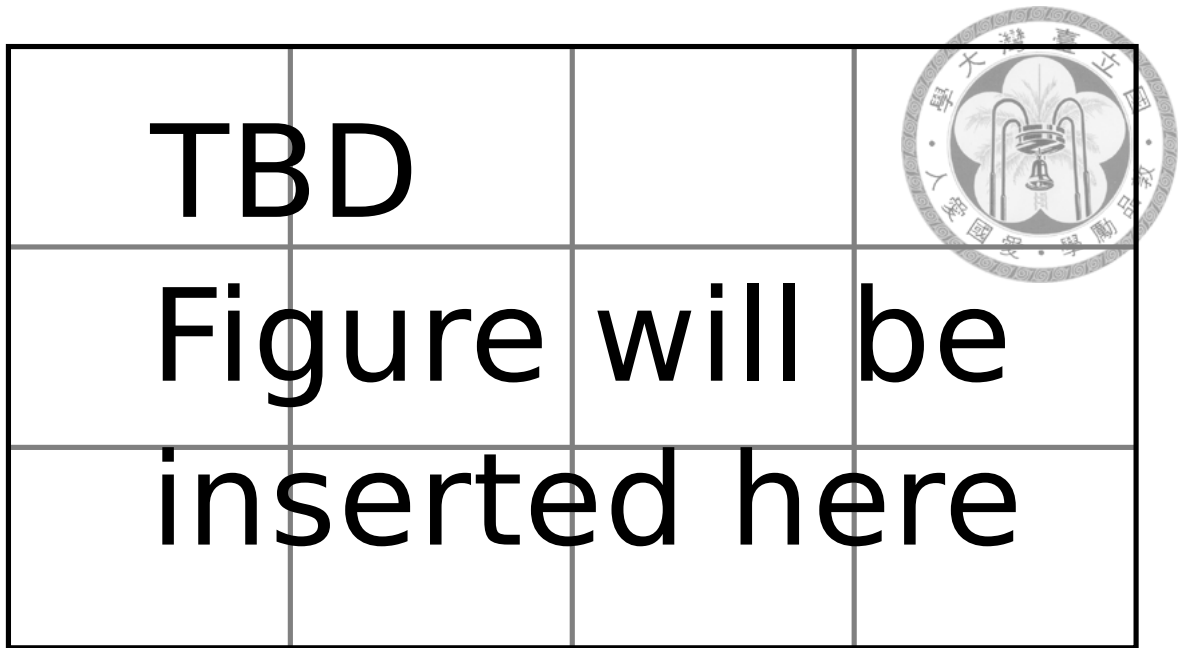


Figure 3.12: The Effect of Automatic Normalization on Onset Deviation feature

not using any normalization (see Fig. 3.12), the method can produce very low onset deviations, roughly centered around zero, while the result from not using any normalization will show a clear trend of increasing deviations.



Chapter 4

Corpus Preparation

An expressive performance corpus is a set of performance samples. Since this research is based on a supervised learning algorithm, a high-quality corpus is essential to our success. Each sample consists of a score and its corresponding human recording. Some metadata such as structure analysis, harmonic analysis etc. may also be included. In this chapter, we will review some the existing corpora, specifications and formats of our corpus, and how we actually construct it.

4.1 Existing Corpora

Unlike other research fields like speech processing or natural language processing, there exist virtually no public accessible corpus for computer expressive performance. CrestMusePEDB [60] (PEDB stands for “Performance Expression Database”) is a corpus created by Japan Science and Technology Agency's CREST program, however, until the time of this writing, we can't establish any contact with the database administrators. They use a graphical interface to annotate the expressive performance parameters from audio recordings. The repertoire covers many piano works from well-known classical composers like Bach, Mozart, and Chopin, and the recordings are from famous pianists. From their website [60] they claim to contain the following data: PEDB-SCR - score text information, PEDB-DEV - performance deviation data and PEDB-IDX - audio performance credit. But the quality of the data is unknown.

Another example is the Magaloff Project [61], which is a joint effort of some universities in Austria. They invite Russian pianist Nikita Magaloff to record all solo works for piano by Frederic Chopin on a Bösendorfer SE computer-controlled grand piano. This corpus became the material for many subsequent researches [62--68]. Flossmann et al., one of the leading researchers of the project, also won the 2008 RenCon contest with a expressive performance system call YQX [69] based on this corpus. However, the corpus is not opened up to the public.

Since both corpora are not available, we need to implement our own one. We will start by defining the specification.

4.2 Corpus Specification

The corpus we need must fulfill the following constraints:

1. All the samples are monophonic, containing only a single melody without chords.
2. No human error, such as insertion, deletion, or wrong pitch exist in the recording; the score and recording are matched note-to-note.
3. Phrasing is annotated by human.
4. The score, recording and phrasing data are in machine-readable format.

Some useful information are excluded because they are less relevant to our system.

Examples are:

1. Advanced Structural Analysis, such as GTTM (Generative Theory of Tonal Music) [70]
2. Harmonic Analysis
3. Instrument specific techniques, such as violin pizzicato, tapping, or bow techniques.
4. Piano paddle usage

Table 4.1: Clementi's Sonatinas Op. 36

Title	Movement	Time Signature
No. 1 Sonatina in C major	I. Allegro	4/4
	II. Andante	3/4
	III. Vivace	3/8
No. 2 Sonatina in G major	I. Allegretto	2/4
	II. Allegretto	3/4
	III. Allegro	3/8
No. 3 Sonatina in C major	I. Spiritoso	4/4
	II. Un poco adagio	2/2
	III. Allegro	2/4
No. 4 Sonatina in F major	I. Con spirito	3/4
	II. Andante con espressione	2/4
	III. Rondó: Allegro vivace	2/4
No. 5 Sonatina in G major	I. Presto	2/2
	II. Allegretto moderato	3/8
	III. Rondó: Allegro molto	2/4
No. 6 Sonatina in D major	I. Allegro con spirito	4/4
	II. Allegretto	6/8



- Other instrument specific instructions, such as piano fingering, violin bow techniques etc.

We choose Clementi's Sonatina Op. 36 for our corpus, because it is a must-learn repertoire for every piano student (especially in Asia), so it's easy to find a wide skill range of performers to record the corpus. These sonatina are in classical style, so the learned model be easily extended to other classical era works like Mozart or Haydn. There are six sonatinas included in Op. 36, the first five have three movements each, and the last one has two movements. The titles, tempo markers and time signatures of all the pieces are listed in Table 4.1

To represent Clementi's work in digital format, we choose MusicXML. MusicXML is a digital score notation using XML (eXtensible Markup Language), it can express most traditional music notations and metadata. Most music notation software and software tool supports musicXML format. An example snippet of a musicXML score is shown in Fig. 4.1 Although MIDI is also a popular candidate for score representation in computer music research, it is designed to hold instrument control signal rather than notation. Some music symbols may not be available in MIDI. Furthermore, MIDI represents music as a series of

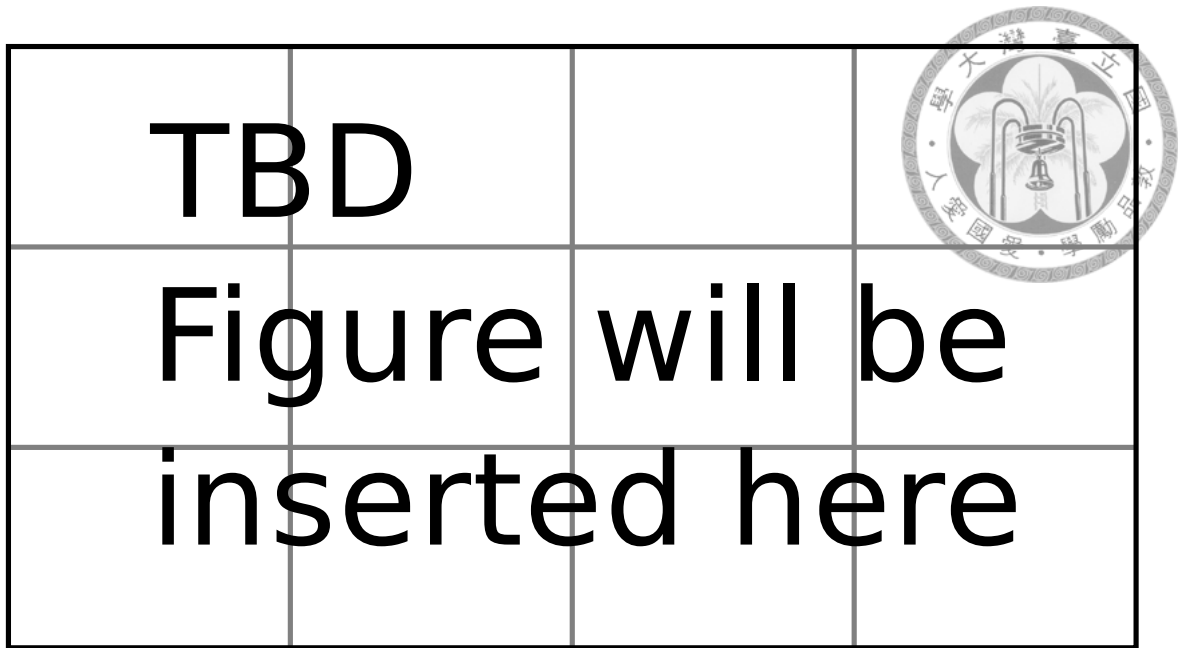


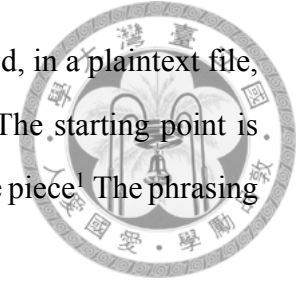
Figure 4.1: Example MusicXML score

note-on and note-off events, which requires additional effort to transform into traditional notation.

But for performance, MIDI is the most suitable format. Using a pressure sensitive digital piano, pianist can record in a natural way. The recordings have high precision in time and pitch, and polyphonic tracks can easily be separated. Although WAV (Waveform Audio Format) audio recording has higher fidelity than MIDI, but they are harder to parse by computers. Without robust onset detection, pitch detection, and source separation, the information is extremely difficult to extract. Manually annotate each WAV recording takes a lot of manpower, and the accuracy may not be consistent.

There's a way to keep both the score and the recording in one single MIDI file. Instead of recording the actual note-on and note-off timing, we keep the nominal note-on and note-off the same as in score. Then, MIDI tempo-change events are inserted before each note to shift the actual timing of the recorded notes. But since MIDI is so limited to represent a score, and it requires complex calculations to recover the performance from fixed notes and tempo-change events, this method is not used in the research.

Finally, we store the phrasing, which is the only metadata we used, in a plaintext file, each line in the phrasing file is the starting point of each phrase. The starting point is defined as the onset timing (in quarter notes) from the beginning of the piece¹. The phrasing is assigned by the us using the following criteria:



1. Phrase may be separated by a salient pause.
2. Phrase may end with a cadence.
3. Phrase may be separated by dramatic change in tempo, key or loudness
4. Repeated structures in tempo or pitch may be a phrase.

The above criteria are just general principles, not strict rules, the phrasing is still decided subjectively.

An automatic phrasing algorithm may be achieved in the future, either by fuzzy rules or by machine learning. With the automatic phrasing capability, the system can become fully automatic. But phrasing controls the structural expression of a piece, we left the phrasing decision to the user. Because we intend to leave some degree of freedom for users to expressive themselves. But note-level expression is too trivial to be assigned by hand, but deciding phrase-level expression is less demanding for an ordinary user.

4.3 Implementation

4.3.1 Score Preparation

The digital scores are downloaded from KernScore website [71]. The scores are transformed into MusicXML from th original Hundrum file format (.krm) using the music21 toolkit. Because this research focus on monophonic melody only, the accompaniments are remove and the chords are reduced to their highest-pitched note, which is usually the most

¹For a phrase that start at a point which is a circulating decimal, for example $\frac{1}{3}$, the starting point can be alternatively defined as any finite decimal between the end of the last phrase and the start of the current phrase. For example, if the last phrase stops at beat 1, the second phrase start at $2\frac{1}{3} = 2.333 \dots$ beat, the start point of the second phrase can be written as 2.3 or 2.0, etc.

salient melody line. The reduced scores are doubled-checked against a printed version publish by Durand & Cie.,Paris [72] to eliminate all errors.



4.3.2 MIDI Recording

We have implemented two methods for recording: First, using a Yamaha digital piano to record MIDI. Second, by tapping on a touch sensitive device to express tempo, duration and loudness. Due to accuracy consideration, only the recordings from Yamaha digital piano are selected.

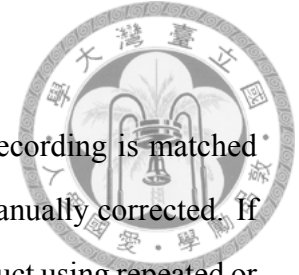
We used a Yamaha P80 88-key graded hammer effect²digital piano for recording. Using a MIDI-to-USB converter, the keyboard is connected to Rosegarden Digital Audio Workstation (DAW) software on a Linux computer. The Rosegarden DAW also generated the metronome sound to help the performer maintain a steady speed. Metronome is mandatory because if the performer plays freely, the tempo information written in the MIDI file will be invalid, which makes subsequent parsing and linear scaling very difficult. So the performers are asked to follow the speed of the metronome, but they can apply any level of rubato they like.

The second method, which is not used in the final experiments, is to utilize touch-enabled input device like smartphone touchscreen or laptop touchpad. We have implemented an prototype using a Synaptics Touchpad on a Lenovo ThinkPad X200i laptop. When the user taps the touchpad, one note from a predefined score will be played, the duration and loudness will be controlled by the time and pressure of the tap. So the user can “play” the touchpad like a musical instrument. This idea has already be used in musical games like Magic Piano [?]. This method requires minimal instrument skill and utilize widely available hardware. But the accuracy in pressure is not satisfying, because most touchpad use contact area to estimate pressure. But it is indeed a low cost alternative to a MIDI digital piano.

²Graded Hammer Effect feature provides realistic key pressure response similar to a traditional acoustic piano

4.3.3 MIDI Cleaning and Phrase Splitting

After the MIDI files are recorded, we use scripts to check if each recording is matched note-to-note with its corresponding score. If not, the mistakes are manually corrected. If there are a small segments that is totally messed up, it will be reconstruct using repeated or similar segments from the same piece. The matched score and MIDI pair are then split into phrases according to the phrasing file. The split phrases are checked again for note-to-note match before they are used.



4.4 Results

Six graduate students (not majored in music) with varying piano skill are invited to record the samples. The number of mistakes they made are listed in Table 4.2.³ These mistakes are identified using the unix `diff` [?] tool. Five of them (A to E) finished Clementi's entire Op. 36, performer F only recorded part of the work. The total number of recordings and the corresponding phrases/notes counts are shown in Table 4.3. TODO:mistakes

The number of phrases (according to our phrasing) and notes are shown in Table 4.4. Fig. 4.2 shows the length distribution of each movement, most movements are around a few hundred notes, except the long No.6 and some short second movements. Fig. 4.3 shows the length distribution in numbers of phrases, most movements are around 20 phrases. The length distribution of the phrases in all six pieces are shown in Fig. 4.4, most phrases are shorter than 30 notes. Some very long phrases exists, they are often virtuoso segments with very fast note sequences, so they are hard to be further split.

TODO:mention Fig. 4.5 TODO:error count

REVIEW1

³The performers are allowed to re-record as many time as they want, so the actual number of mistakes may be higher.

Table 4.2: Number of Mistakes in Corpus

Performer	1-1	1-2	1-3	2-1	2-2	2-3	3-1	3-2	3-3	4-1	4-2	4-3	5-1	5-2	5-3	6-1	6-2	Subtotal
A	0	5	2	4	3	0	4	2	2	4	5	9	9	2	3	4	1	59
B	2	1	1	2	2	1	6	0	3	2	3	6	12	3	3	10	7	64
C	1	1	0	1	0	1	2	0	0	3	2	3	10	1	35	6	1	67
D	0	1	1	2	3	1	4	1	1	10	6	3	10	2	7	13	2	67
E	2	3	4	4	0	3	4	0	0	21	6	22	23	3	9	18	13	135
F	1	3	2	11	6	8	7	2	6		15				20			81
Subtotal	6	14	10	24	14	14	27	5	12	40	37	43	64	11	77	51	24	473



Table 4.3: Total Recorded Phrases and Notes Count

Title	Recordings Count	Total Phrases	Total Notes
No. 1 Mov. I	6	72	1332
No. 1 Mov. II	6	60	882
No. 1 Mov. III	6	102	1566
No. 2 Mov. I	6	108	1920
No. 2 Mov. II	6	36	750
No. 2 Mov. III	6	168	2484
No. 3 Mov. I	6	156	3156
No. 3 Mov. II	6	42	444
No. 3 Mov. III	6	120	2628
No. 4 Mov. I	5	80	2325
No. 4 Mov. II	6	78	1332
No. 4 Mov. III	5	85	1920
No. 5 Mov. I	5	85	3360
No. 5 Mov. II	5	70	1580
No. 5 Mov. III	6	144	3384
No. 6 Mov. I	5	145	4180
No. 6 Mov. II	6	78	2754
Total	97	1629	35997



Table 4.4: Phrases and Notes Count for Clementi's Sonatina Op. 36

Title	Phrases Count	Notes Count
No. 1 Mov. I	12	222
No. 1 Mov. II	10	147
No. 1 Mov. III	16	261
No. 2 Mov. I	18	320
No. 2 Mov. II	6	125
No. 2 Mov. III	28	414
No. 3 Mov. I	25	526
No. 3 Mov. II	6	74
No. 3 Mov. III	19	438
No. 4 Mov. I	25	465
No. 4 Mov. II	12	222
No. 4 Mov. III	16	384
No. 5 Mov. I	17	672
No. 5 Mov. II	13	316
No. 5 Mov. III	24	564
No. 6 Mov. I	28	836
No. 6 Mov. II	11	459
Total	286	6445

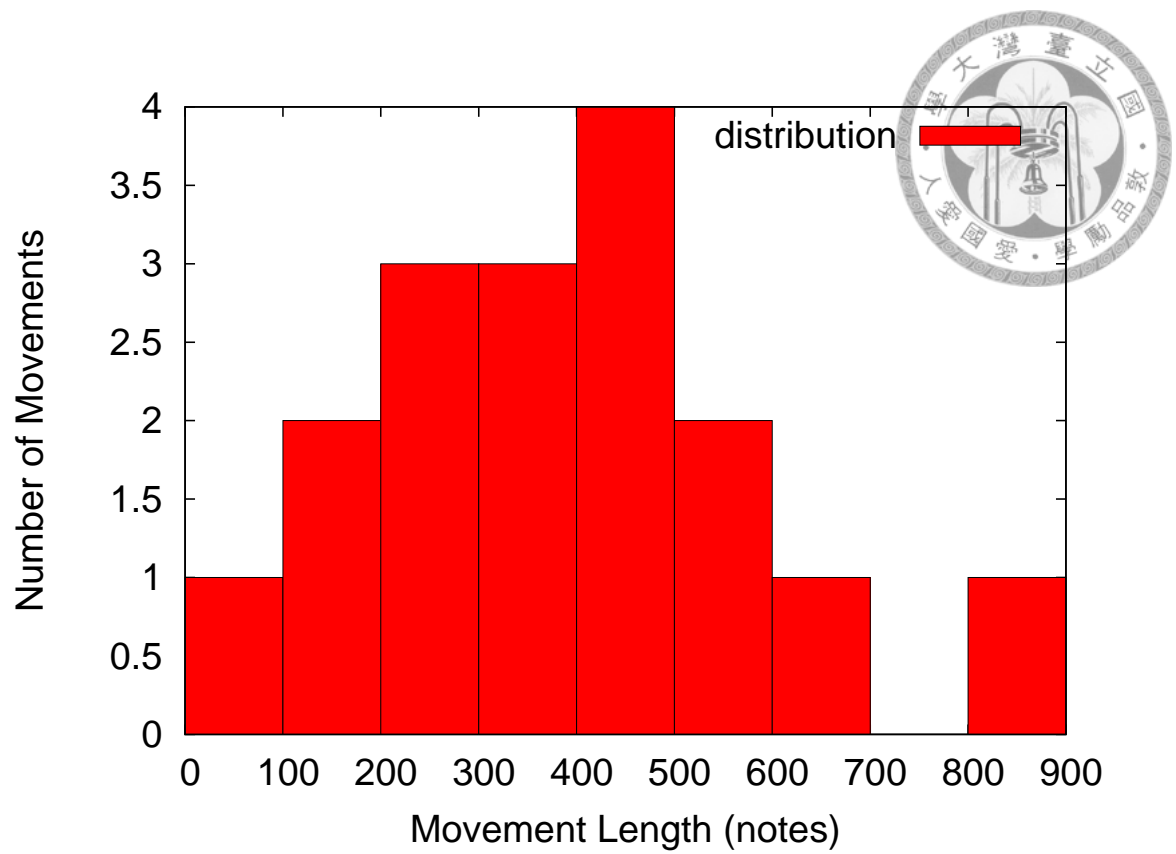


Figure 4.2: Movements Length (in Notes) Distribution

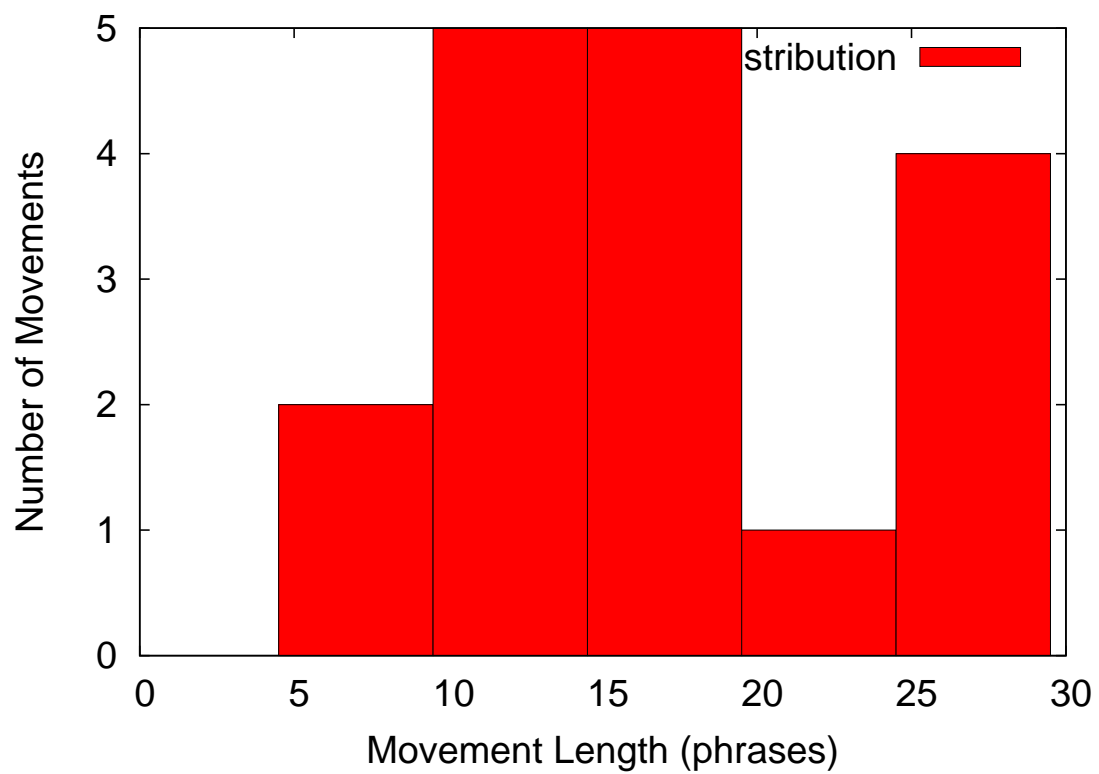


Figure 4.3: Movements Length (in Phrases) Distribution

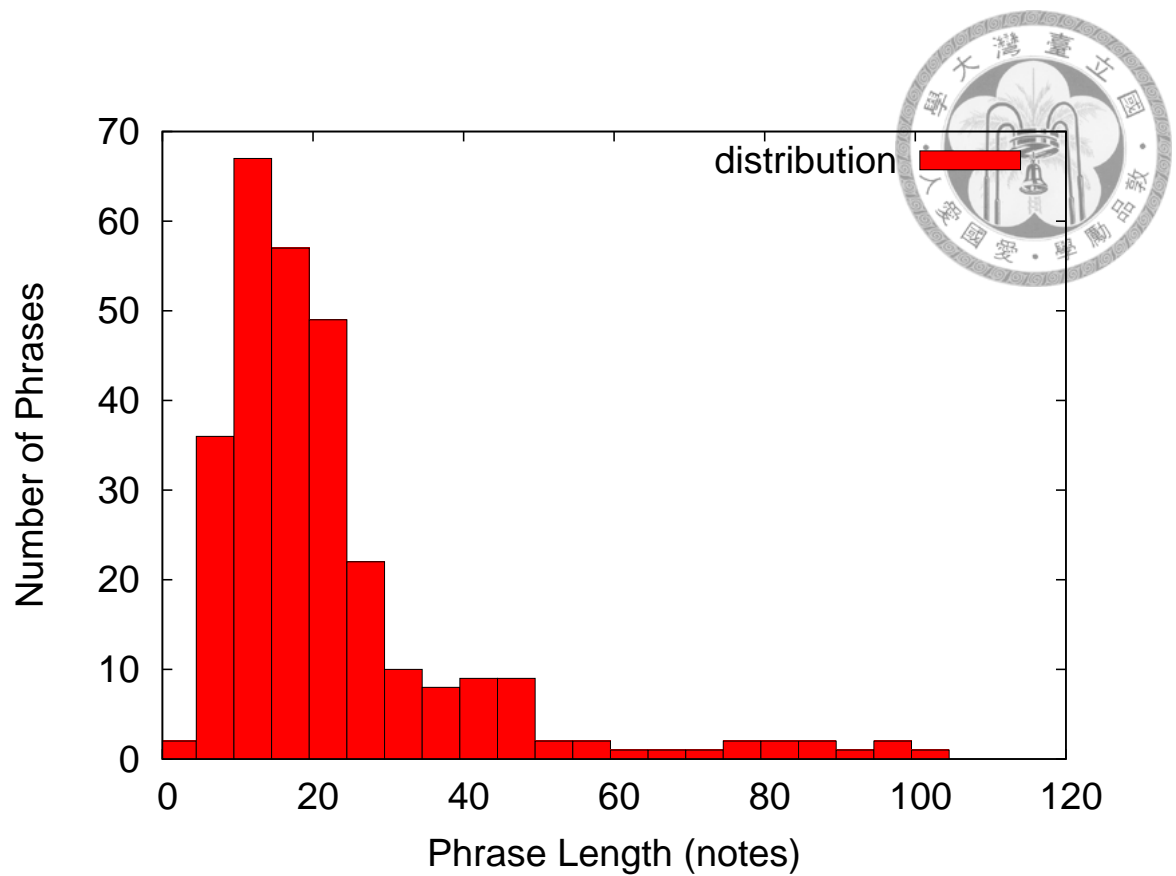


Figure 4.4: Phrase Length (in Notes) Distribution

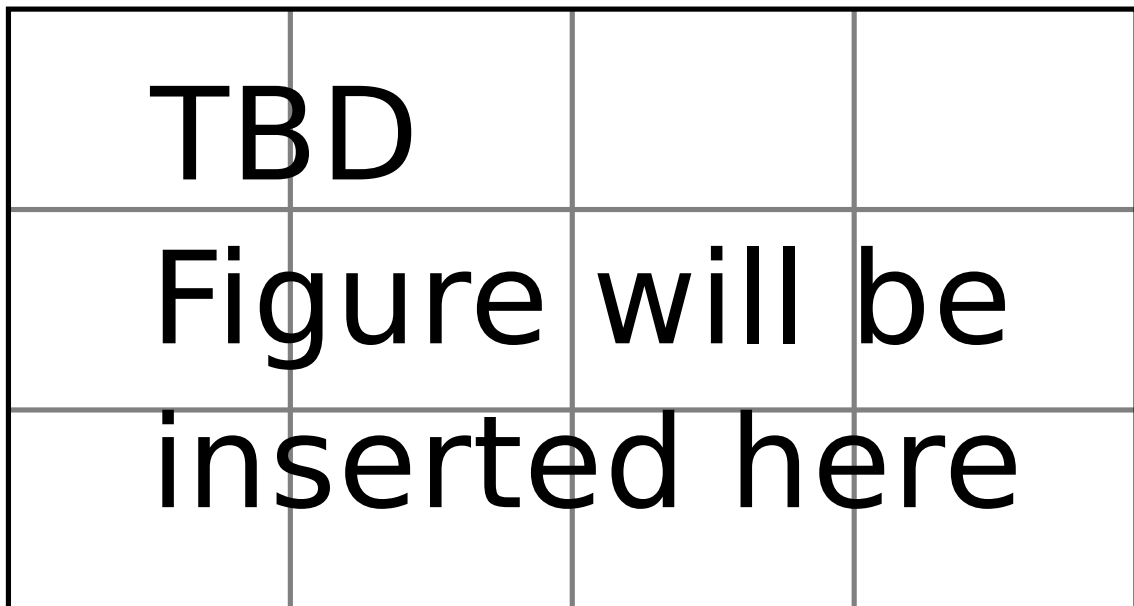


Figure 4.5: Example Recording Compared to Score (Pianoroll)



Chapter 5

Experiments, Results and Discussions

REVIEW1 **TODO:general introduction to the experiments** In this chapter, we will use some experiment results to demonstrate the effectiveness of our method. Section 5.1 deals with the onset deviation problem highlighted in Section 3.5.3. Section 5.2 discusses how the various parameters in our system is decided, and their effect on the model and computing resource required. Section 5.3 describes a Turing test to see if audience can or can't identify the difference between generated and human performances.

5.1 Onset Deviation Normalization

TODO:onset deviation problem review As mentioned in Section 3.5.3, a bad normalization method will usually result in unreasonable high onset deviation, which will make the expressive performance sound very unnatural. To overcome this challenge, we proposed a automatic way to select the normalization parameter. In this section, we will evaluate the effectiveness of the method.

We extract the onset deviation feature from a single performer's recording, using the four types of normalization method mentioned before and the automatic normalization method. The extracted onset deviations are shown in Fig. 5.1 to Fig.5.3. Each dotted line from left to right represents a phrase in the corpus. Each dot represents the onset deviation value of a note. The notes are spread uniformly on the horizontal axis, which only shows the order of appearance, not the real time scale. First note that aligning the end of the last

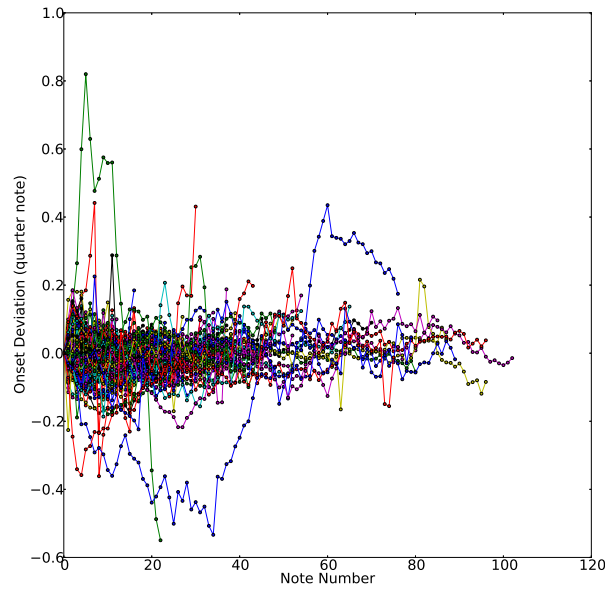


Figure 5.1: Onset Deviations Using Auto Normalization Method

notes results in very large deviation in a few phrases. This is because performers tend to extend the last note in certain phrases to emphasize the end of a phrase or section. This kind of extension will cause the last notes onset to be far apart. So clearly this method is not a good way. Fig. 5.1 and Fig. 5.2 looks much similar, but the maximum and minimum value in Fig. 5.2 is more dramatic than those in Fig. 5.1, which shows that the automatic normalization method can indeed find a better fit between the recorded phrase and the score. Another benefit of the automatic normalization is that the last note onset is not force aligned, which allows more free expression on the last note. In Fig. 5.2, the end of a line, i.e. the last note, always goes back to zero; however in Fig. 5.1, the end of a line can end in a wide range of values.

5.2 Parameter Selection

5.2.1 SVM-related Parameters

Since SVM-HMM is a combination of SVM and HMM, there are many parameters which needs adjustment from both model. Two parameters are tested to find their opti-

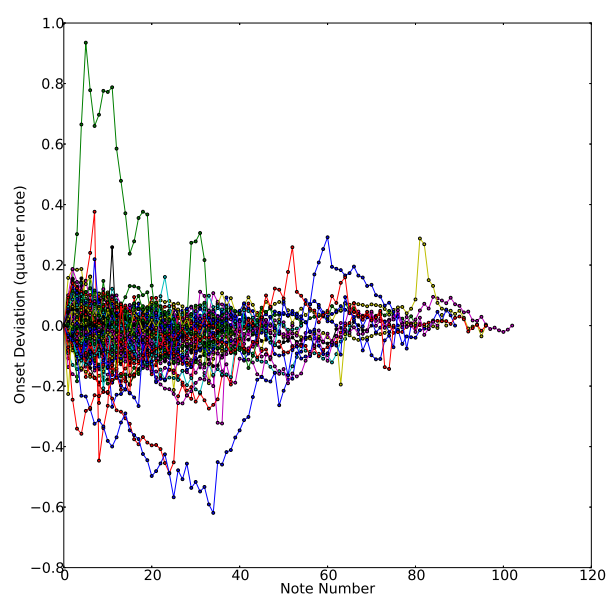


Figure 5.2: Onset Deviations By Aligning Last Notes Onset

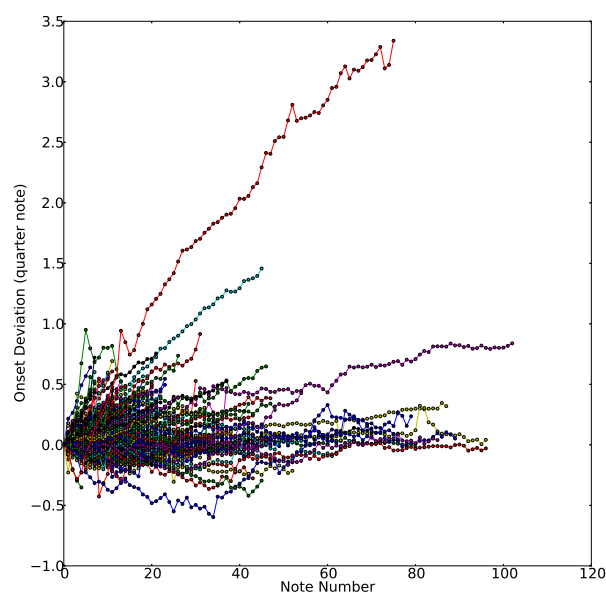


Figure 5.3: Onset Deviations By Aligning Last Notes Off

mal value: the termination accuracy ε and the misclassification penalty factor C in SVM. Because the algorithm is an iterative one, the ε parameter defines the desired accuracy required for the algorithm to terminate. A smaller ε will result in higher accuracy, but increased execution time (because of more iterations run.) The C parameter determines how hard non-separable samples should be penalised. A large C will sacrifice larger margin for lower misclassification error, but it will make the execution longer.

Because it is desirable to generate a performance in a style similar to the training examples, we use the whole set of Clementi's Sonatinas Op.36 from a single performer, and split them into two sets: the training set includes pieces No. 2 to No. 6, and the testing set includes piece No. 1. We train a model with the training set, and use the learned model to generate expressive performance. The computer generated expressive performance should have a similar expression to the testing set.

To measure the effectiveness of the ε and C parameters, the generated performance is compared to the performance recorded by the performer. Ideally, the generated performance will be very similar (in expression) to the recording. So, for every pair of the generated and recorded performances, we calculate their distance, and take the median value of all the distances for every C . Note that each performance feature has its own model, so we will be looking at a single performance feature and its C parameter at a time. First, the generated performance features sequence and the recorded one are normalized to a range from 0 to 1. The normalization is required because we want to tolerate linear scaling. Then the Euclidean distance of the two normalized sequence is calculated and divided by the length (in notes) of the phrase, since the phrase can have arbitrary length.

First we will fix C at 0.1 and test different ε 's: 100, 10, 1, 0.75, 0.5 and 0.1. Then, we fix ε at the optimal value determined in the previous step and test ε 's: 10^{-3} , 10^{-2} , 10^{-1} , 0.5, 1, 5, with other parameters set to default. And we will evaluate the optimal parameters and evaluate their execution time. For each ε or C , we calculate the distance between the generated pieces and recorded examples for all phrases in the testing set for each performer. And we take the median of all these distances for each ε or C .

TODO: median The performance regeneration accuracy for various ε 's are shown

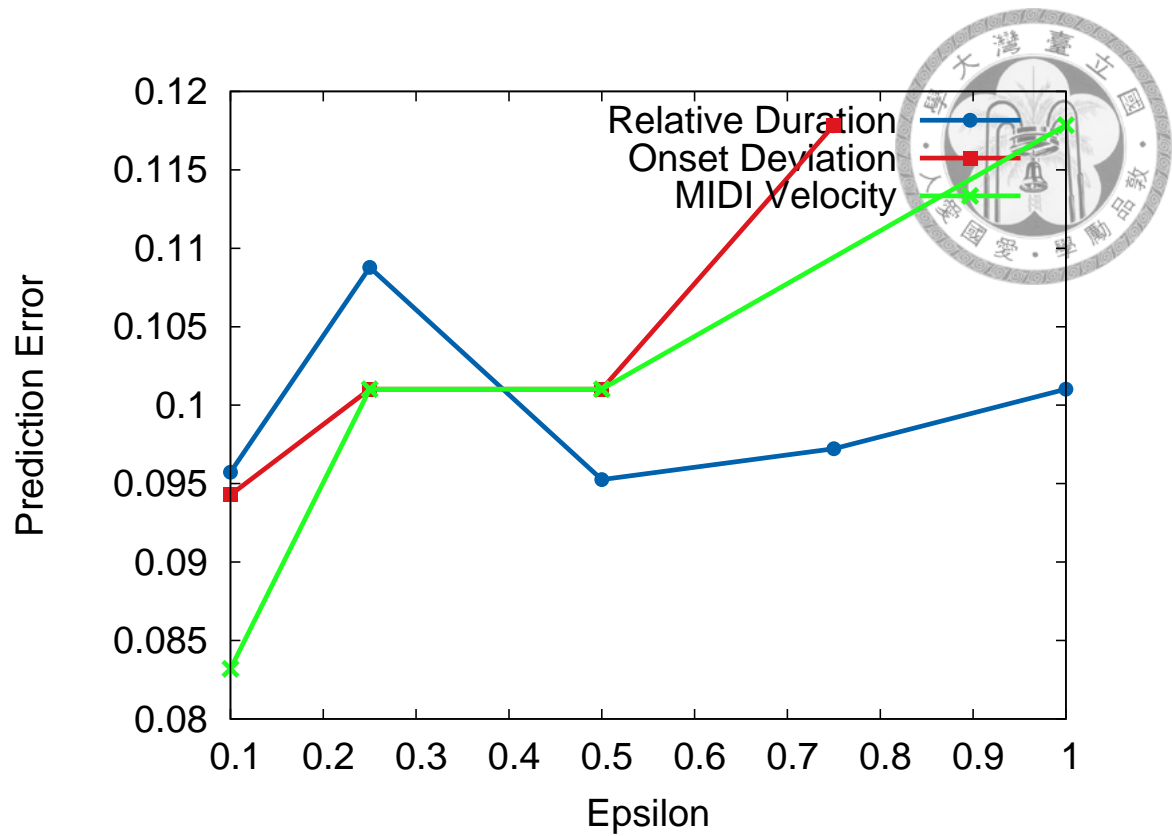


Figure 5.4: Accuracy for Different ε 's

in Fig. 5.4. And the time for various ε 's are shown in Fig. 5.5. For ε value 100 and 10, the termination criteria is too generous, so the learning algorithm terminates almost immediately. Therefore, the model hardly learns anything, the output is a fixed value for any input. So we abandon the data points because the model had learned nothing. We can see that the accuracy drops slowly when ε becomes smaller. But after ε is smaller than 0.5, the accuracy doesn't drop anymore. So we will choose $\varepsilon = 0.5$ for the rest of the experiment to avoid unnecessary computations.

As for different C parameter, the accuracy and execution time are shown in Fig.5.6 and Fig. 5.7 respectively. The accuracy forms a U shape curve, with the valley near 0.1. But the execution time grows as C goes larger, so we should choose $C = 0.1$ as our optimal C.

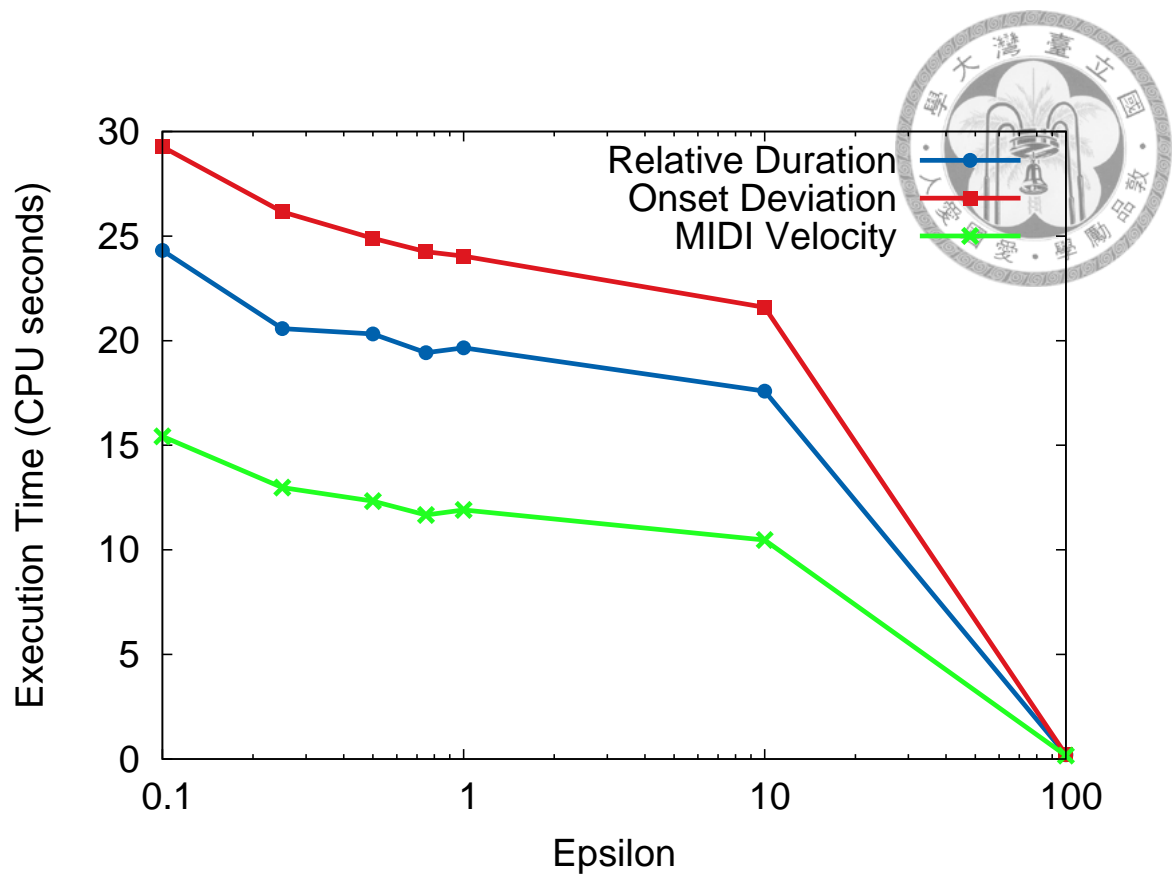


Figure 5.5: Execution Time for Different ϵ 's

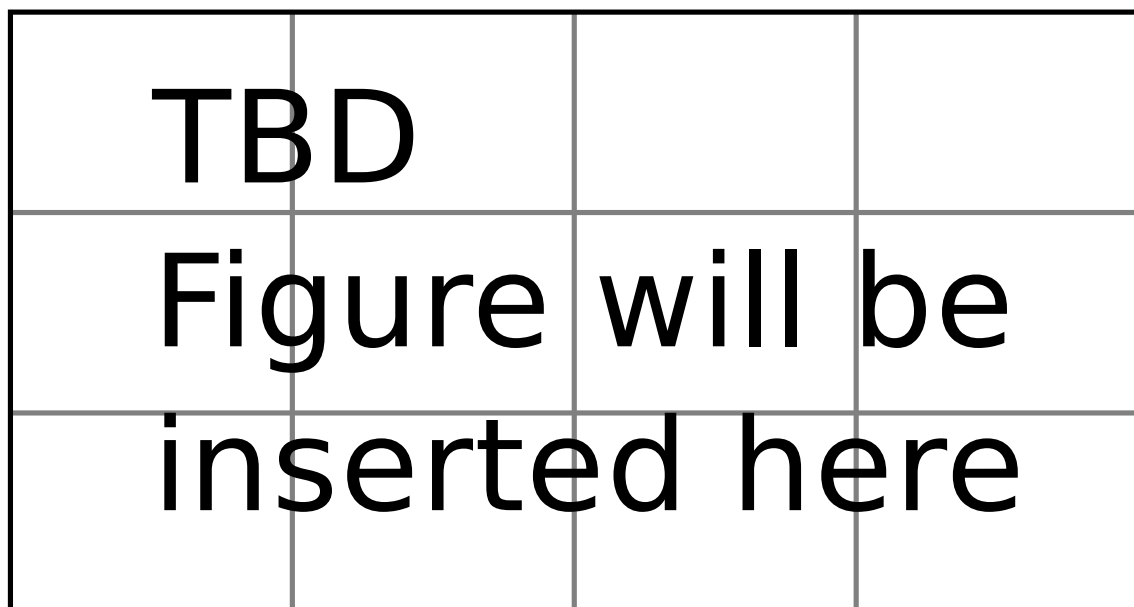


Figure 5.6: Accuracy for Different C's

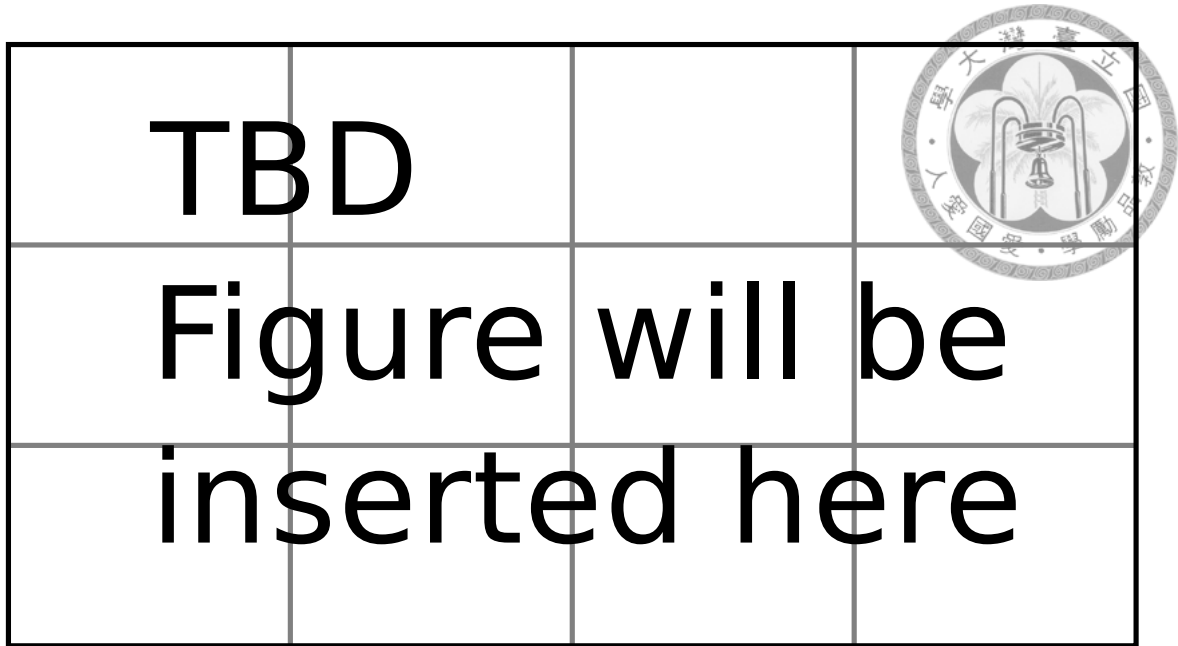


Figure 5.7: Execution Time for Different C's

5.2.2 Quantization Parameter

Besides ε and C , the number of quantization levels for SVM-HMM input is also has some impact on the execution time. If the performance features are quantized into more fine-grained levels, the quantization errors can be reduced, but the execution time and memory usage will grow dramatically. Also, larger number of intervals doesn't imply more accurate or robust model. Because SVM-HMM is originally used in part-of-speech tagging problem, if we use divide the performance features into more intervals, than there will be fewer samples for each interval. But it is more desirable that similar phrases will be quantized to the same bin, instead of split them into more fine-grain bins. For example, if a three note segment is played once in the following MIDI velocity: (60, 70, 80), and the same phrase is played again in (60.1, 69.9, 80.1). If we have a quantization interval width of, say, 0.05, then 60 and 60.1 may be quantized into different bins, and 70 and 69.9 may also be quantized to different bins, so the two phrases will be considered as two different case. However, if the quantization interval width is 1, both phrases may be the same after quantization, which is more desirable because the SVM-HMM algorithm can capture the similarity in the two samples.

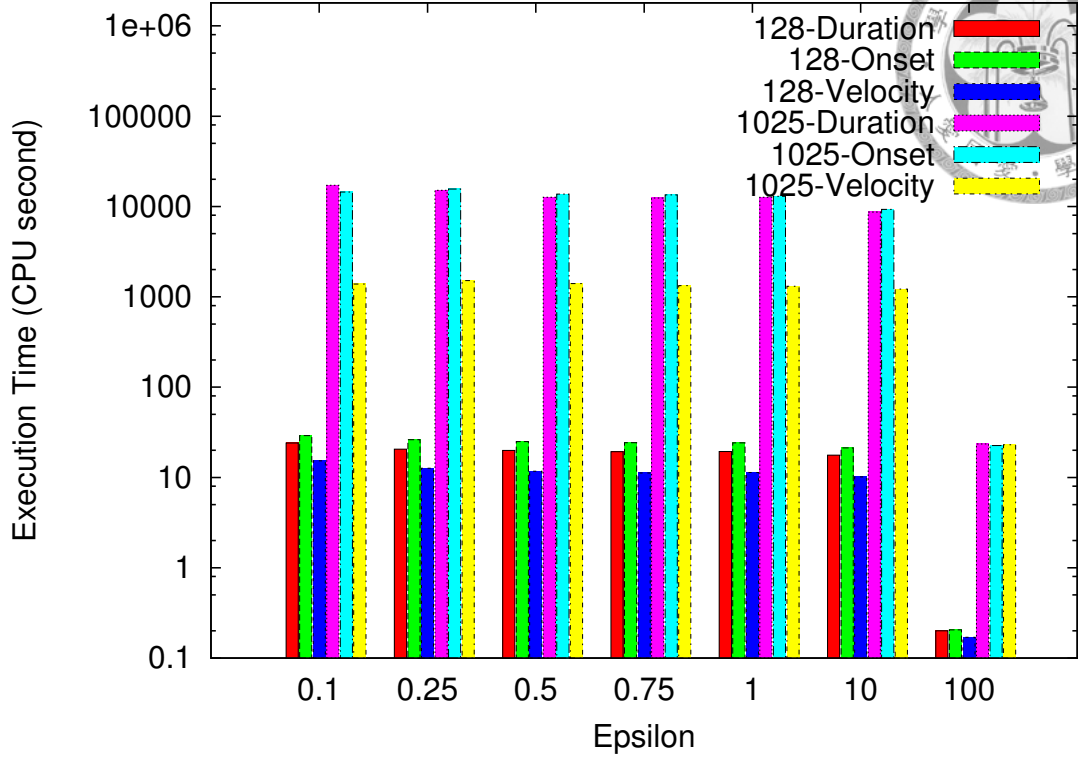


Figure 5.8: Execution Time for different Quantization Granularity

Initially, we tried to quantized the values into 1025 uniform width bins, wishing to minimize quantization error. But it take very long time to learn a model. So we switch to a more reasonable value: 128. From empirical data, onset deviation feature rarely exceeds ± 1 , so the quantization interval is around $\frac{1-(-1)}{128} = 0.015625$. Most duration ratios falls between zero and three, so the interval is $\frac{3-0}{128} = 0.0234375$. And MIDI velocity is around 30 to 90, so the interval is about $\frac{90-30}{128} = 0.46875$. This level of granularity is good enough for our performance system, and can dramatically reduce the execution time without sacrificing the expressiveness of the models.

We repeat the ε selection experiment in the previous section for quantization level of 1025 and 128. The execution time (in CPU second) is shown in Fig. 5.8. The time required for 1025 is larger than 128 by orders of magnitudes. The outcome, however, is still expressive, as can be heard in the Section 5.3.

5.3 Human-like Performance

The most general purpose of a expressive performance system is to create expressive, non-robotic music as oppose to deadpan MIDI. Therefore, we would like to perform a Turing-test-style survey to find out how people think about the our generated expressive music.

In this survey, 1518 computer generated expressive phrases and their corresponding human recording are used as candidates. Each test subject was given 10 randomly selected computer generated phrase and 10 random human recordings, the 20 phrases are presented in random order. He/She was asked to evaluate each phrase according to the following criteria, which are proposed by RenCon contest [?]:

1. Technical control – if a performance sounds like it is technically skilled thus performed with accurate and secure notes, rhythms, tempo and articulation.
2. Humanness – if the performance sounds like a human was playing it.
3. Musicality – how musical the performance is in terms of tone and color, phrasing, flow, mood and emotions
4. Expressive variation – how much expressive variation (versus deadpan) there is in the performance.

In RenCon, each judge is asked to give separate ratings for each criteria. But we believe that the understanding the difference between the four criteria and give separate rating is too demanding for less-experienced participant, so we asked test subjects to vote an overall rating from one to five. One being very bad, five being very good. The test subject are also asked to report their musical proficiency in a three level scale:

1. No experience in music
2. Amateur performer
3. Professional musician, musicologist or student majored in music



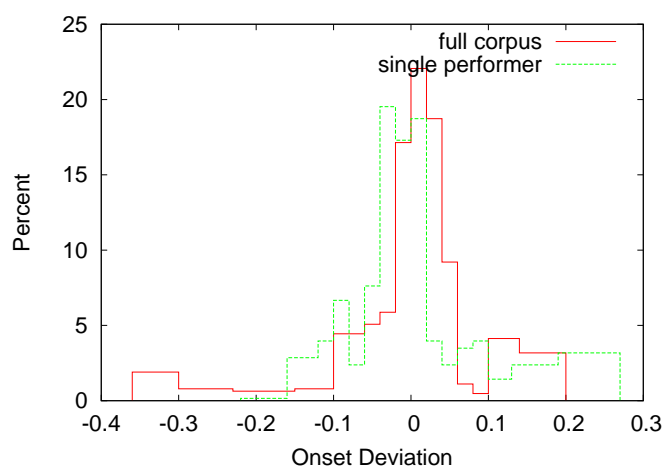


Figure 5.9: Distribution of Onset Deviation Values from Full Corpus versus Single Performer's Corpus

To generate the expressive performance phrase. We follow a six-fold cross validation pattern. For each performer in the corpus, we use all his/her recorded phrase from Clementi's Op.36 No. 2 to No. 6 to train a model. Then the model is used to generate all phrases from Clementi's Op.36 No. 1. The generate phrases and the performer's recording of piece No. 1 will all be included into the test set. Maybe we should compare to other people's recording?

The process is repeated, but this time we use pieces No. 1, 3, 4, 5, 6 to train a model, and generate piece No. 2, and so on. So all six pieces will have a computer generated version and recorded version for each player's corpus.

We have also tried to use all performer's recording instead of just one person's corpus to generate models. However, the expressive variation from the model is much smaller than a single performer's model. This is because the expression from different performer may cancel out each other. The distribution of each performance features are presented in Fig. 5.9 to Fig. 5.11. For each performance feature, the frequency of occurrence of different feature values are counted and plotted as histograms. The features generated from the full corpus are more concentrated, which results in less dramatic expression.

TODO:Turing test result and discussion

We received 119 valid samples from the survey. Fifty of them are from people with no music background, 59 are from amateur musicians, and the rest 10 are from professional musicians. The average rating given to computer generated performance and human

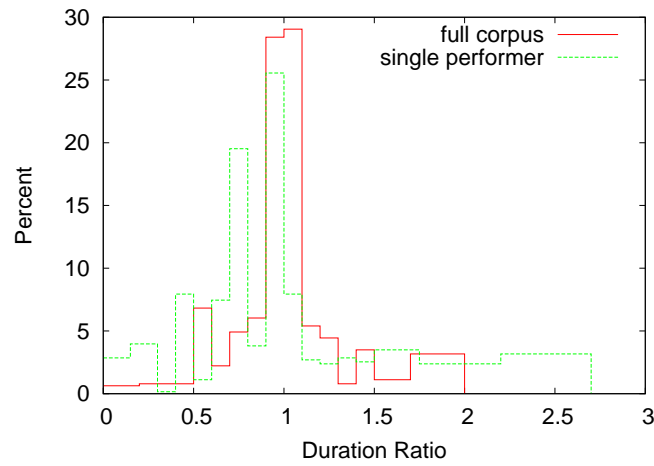


Figure 5.10: Distribution of Duration Ratio Values from Full Corpus versus Single Performer's Corpus

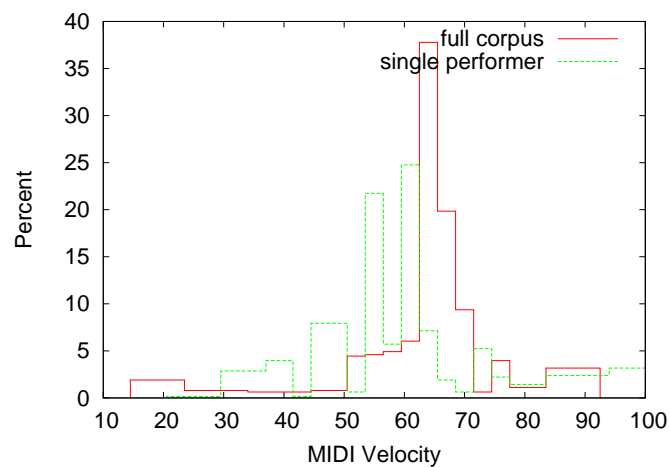


Figure 5.11: Distribution of MIDI Velocity Values from Full Corpus versus Single Performer's Corpus

Table 5.1: Average Rating for Generated Performance and Human Recording

	Computer	Human
No experience	3.243	3.391
Amateur	2.798	3.289
Professional	2.430	3.010
Total	2.952	3.306



Table 5.2: Average Rating for Generated Performance and Human Recording under Different Part of the corpus

	A,B		C-F	
	Computer	Human	Computer	Human
No experience	3.067	3.302	3.363	3.451
Amateur	2.680	3.347	2.863	3.286
Professional	2.048	3.162	2.708	2.921
Total	2.776	3.313	3.066	3.323

recordings are listed in Table 5.1. It is clear that for professional and amateur musician, the average rating given to human performances are higher than computer performances. However, for participants who have no experience in music, the rating is much closer. A Student T-test on the two ratings for participants with no experience yields a p-value of 0.0312, therefore we can't reject the null hypothesis that the two ratings are different under a significance level of 99%.

In order to get more insight from the ratings, we can further divide the performers in the corpus in to two categories based on their piano skill. By the number of mistakes made (Table 4.2), performer A and B are considered more skilled, and performer C, D, E and F are considered less skilled. The average rating given to the two categories are listed in Table 5.2. The distance between computer and human performances are smaller for less skilled group (C to F) than the more skilled group (A and B). This is probably because our system makes some mistakes that are similar to the mistakes made by less-skilled performers. For example, unsteady tempo, sudden change in loudness, hesitation and emphasising notes that are structurally less important are all common problems that exists in both less-skilled performance and computer generated performance. But for more-skilled performer, because of they have better technical control and sense of musical structure, the problems described above will happen less often.

If we look into each individual participant, we can see if a participant gives higher (av-

Table 5.3: Number of Participants Who Gives Higher Rating to Generated Performance, Human Recordings or Equal Rating

	Computer	Equal	Human	Total
No experience	19	7	24	50
Amateur	7	3	49	59
Professional	1	1	8	10
Total	27	11	81	119



Table 5.4: Number of Participants Who Gives Higher Rating to Generated Performance, Human Recordings or Equal Rating under Different Part of the Corpus

	A,B			C-F			Total
	Computer	Equal	Human	Computer	Equal	Human	
No experience	5	4	6	14	3	18	50
Amateur	2	1	18	5	2	31	59
Professional	0	1	3	1	0	5	10
Total Result	7	6	27	20	5	54	119

erage) rating to computer or human performances, or equal ratings. The number of participants who fall into each categories are shown in Table 5.3. Twenty-six non-experienced participants give higher or equal rating to computer than human, slightly higher than twenty-four people who gives higher rating to human. For amateur and professional musicians, the number of people who prefers human are much higher. Table 5.4 is a table similar to 5.3, but split into two categories just like Table 5.2. The results are similar to Table 5.3: the difference between computer and human is higher for skilled performers (A and B) than less-skilled performers (C to F).



Chapter 6

Conclusions

REVIEW1 TODO:summary


There are many room for improvements. Structural expressions such as phrasing, contract between sections, or even contrast between movements can be added. Other information like text notations, harmonic analysis and musicological analysis can be added. Supporting homophonic or polyphonic music is also important for the system to be useful. Sub-note expressions like physical model synthesizer or envelope shaping can also be applied to generate performance system for specific musical instruments. Error model?

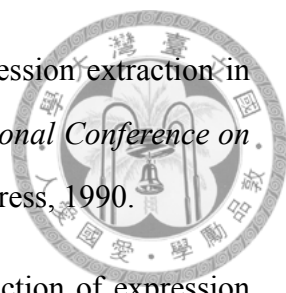
It's also crucial to test the system on more samples from different genre or music style, to verify the effectiveness of different systems. We also believe combining rule-based model and machine learning model may be a possible direction for computer expressive music performance research, because machine learning methods can learn subconscious expressions, while rule based system can enable the user to control the overall expression of a piece of music with ease.

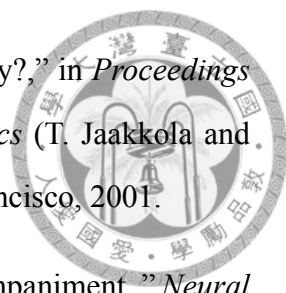


Bibliography

- [1] R. Hiraga, R. Bresin, K. Hirata, and R. KH, “Turing test for musical expression proceedings of international conference on new interfaces for musical expression,” in *Proceedings of 2004 new interfaces for musical expression conference* (Y. Nagashima and M. Lyons, eds.), (Hamatsu, Japan), pp. 120--123, ACM Press, 2004.
- [2] A. Kirke and E. R. Miranda, “An Overview of Computer Systems for Expressive Music Performance,” in *Guide to Computing for Expressive Music Performance* (A. Kirke and E. R. Miranda, eds.), pp. 1--47, Springer, 2013.
- [3] “Sibelius.” <http://www.avid.com/us/products/sibelius/pc/Play-perform-and-share>.
- [4] “Rachmianinoff - Plays Rachmaninoff.” <https://www.zenph.com/rachmaninoff-plays-rachmaninoff>, 2009.
- [5] A. Friberg, R. Bresin, and J. Sundberg, “Overview of the KTH rule system for musical performance,” *Advances in Cognitive Psychology*, vol. 2, pp. 145--161, Jan. 2006.
- [6] W. A. Sethares, *Tuning, Timbre, Spectrum, Scale*. Springer, 2005.
- [7] M. Hashida, N. Nagata, and H. Katayose, “Pop-E: a performance rendering system for the ensemble music that considered group expression,” in *Proceedings of 9th International Conference on Music Perception and Cognition* (M. Baroni, R. Addressi, R. Caterina, and M. Costa, eds.), (Bologna, Spain), pp. 526--534, ICMPC, 2006.

- 
- [8] S. R. Livingstone, R. Mühlberger, A. R. Brown, and A. Loch, “Controlling musical emotionality: an affective computational architecture for influencing musical emotions,” *Digital Creativity*, vol. 18, pp. 43--53, Mar. 2007.
 - [9] N. P. M. Todd, “A computational model of rubato,” *Contemporary Music Review*, vol. 3, pp. 69--88, Jan. 1989.
 - [10] N. P. McAngus Todd, “The dynamics of dynamics: A model of musical expression,” *The Journal of the Acoustical Society of America*, vol. 91, p. 3540, June 1992.
 - [11] N. P. M. Todd, “The kinematics of musical expression,” *The Journal of the Acoustical Society of America*, vol. 97, p. 1940, Mar. 1995.
 - [12] M. Clynes, “Generative principles of musical thought: Integration of microstructure with structure,” *Journal For The Integrated Study Of Artificial Intelligence*, 1986.
 - [13] M. Clynes, “Microstructural musical linguistics: composers' pulses are liked most by the best musicians,” *Cognition*, 1995.
 - [14] M. Johnson, “Toward an expert system for expressive musical performance,” *Computer*, vol. 24, pp. 30--34, July 1991.
 - [15] R. B. Dannenberg and I. Derenyi, “Combining instrument and performance models for high-quality music synthesis,” *Journal of New Music Research*, vol. 27, pp. 211-238, Sept. 1998.
 - [16] R. B. Dannenberg, H. Pellerin, and I. Derenyi, “A Study of Trumpet Envelopes,” in *Proceedings of the 1998 international computer music conference* (O. 1998, ed.), (Ann Arbor, Michigan), pp. 57--61, International Computer Music Association, 1998.
 - [17] G. Mazzola and O. Zahorka, “Tempo curves revisited: Hierarchies of performance fields,” *Computer Music Journal*, vol. 18, no. 1, pp. 40--52, 1994.
 - [18] G. Mazzola, *The Topos of Music: Geometric Logic of Concepts, Theory, and Performance*. Basel/Boston: Birkhäuser, 2002.


- 
- [19] H. Katayose, T. Fukuoka, K. Takami, and S. Inokuchi, "Expression extraction in virtuoso music performances," in *Proceedings of 10th International Conference on Pattern Recognition*, vol. i, pp. 780--784, IEEE Comput. Soc. Press, 1990.
 - [20] H. Katayose, T. Fukuoka, K. Takami, and S. Inokuchi, "Extraction of expression parameters with multiple regression analysis," *Journal of Information Processing Society of Japan*, no. 38, pp. 1473--1481, 1997.
 - [21] O. Ishikawa, Y. Aono, H. Katayose, and S. Inokuchi, "Extraction of Musical Performance Rules Using a Modified Algorithm of Multiple Regression Analysis," in *International Computer Music Conference Proceedings*, (Berlin, Germany), pp. 348--351, International Computer Music Association, San Francisco, 2000.
 - [22] S. Canazza, G. De Poli, C. Drioli, A. Rodà, and A. Vidolin, "Audio Morphing Different Expressive Intentions for Multimedia Systems," *IEEE MultiMedia*, vol. 7, pp. 79--83, July 2000.
 - [23] S. Canazza, A. Vidolin, G. De Poli, C. Drioli, and A. Rodà, "Expressive Morphing for Interactive Performance of Musical Scores," p. 116, Nov. 2001.
 - [24] S. Canazza, G. De Poli, A. Rodà, and A. Vidolin, "An Abstract Control Space for Communication of Sensory Expressive Intentions in Music Performance," *Journal of New Music Research*, vol. 32, pp. 281--294, Sept. 2003.
 - [25] R. Bresin, "Artificial neural networks based models for automatic performance of musical scores," *Journal of New Music Research*, vol. 27, pp. 239--270, Sept. 1998.
 - [26] A. Camurri, R. Dillon, and A. Saron, "An experiment on analysis and synthesis of musical expressivity," in *Proceedings of 13th colloquium on musical informatics*, (L'Aquila, Italy), 2000.
 - [27] G. Grindlay, *Modeling expressive musical performance with Hidden Markov Models*. Phd thesis, University of Santa Cruz, CA, 2005.


- 
- [28] C. Raphael, “Can the computer learn to play music expressively?,” in *Proceedings of the 8th Int. Workshop on Artificial Intelligence and Statistics* (T. Jaakkola and T. Richardson, eds.), pp. 113--120, Morgan Kaufmann, San Francisco, 2001.
- [29] C. Raphael, “A Bayesian Network for Real-Time Musical Accompaniment.,” *Neural Information Processing Systems*, no. 14, pp. 1433--1440, 2001.
- [30] C. Raphael, “Orchestra in a box: A system for real-time musical accompaniment,” in *Proceedings of 2003 International Joint conference on Artificial Intelligence (Working Notes of IJCAI-03 Rencon Workshop)* (G. Gottob and T. Walsh, eds.), (Acapulco, Mexico), pp. 5--10, Morgan Kaufmann, San Francisco, 2003.
- [31] L. Dorard, D. Hardoon, and J. Shawe-Taylor, “Can style be learned? A machine learning approach towards ‘performing’ as famous pianists.,” in *Proceedings of the Music, Brain and Cognition Workshop -- Neural Information Processing Systems*, Whistler, Canada, 2007.
- [32] M. Wright and E. Berdahl, “Towards machine learning of expressive microtiming in Brazilian drumming,” in *Proceedings of the 2006 International Computer Music Conference* (I. Zannos, ed.), (New Orleans, USA), pp. 572--575, ICMA, San Francisco, 2006.
- [33] R. Ramirez and A. Hazan, “Modeling Expressive Music Performance in Jazz.,” in *Proceedings of 18th international Florida Artificial Intelligence Research Society Sonference (AI in Music and Art)*, (Clearwater Beach, FL, USA), pp. 86--91, AAAI Press, Menlo Park, 2005.
- [34] R. Ramirez and A. Hazan, “Inducing a generative expressive performance model using a sequential-covering genetic algorithm,” in *Proceedings of 2007 annual conference on Genetic and evolutionary computation*, (London, UK), ACM Press, New York, 2007.
- [35] Q. Zhang and E. Miranda, “Towards an evolution model of expressive music performance,” in *Proceedings of the 6th International Conference on Intelligent Systems*

Design and Applications (Y. Chen and A. Abraham, eds.), (Jinan, China), pp. 1189-1194, IEEE Computer Society, Washington, DC, 2006.



- [36] E. Miranda, A. Kirke, and Q. Zhang, “Artificial evolution of expressive performance of music: An imitative multi-agent systems approach,” *Computer Music Journal*, vol. 34, no. 1, pp. 80--96, 2010.
- [37] Q. Zhang and E. R. Miranda, “Evolving Expressive Music Performance through Interaction of Artificial Agent Performers,” in *Proceedings of ECAL 2007 workshop on music and artificial life (MusicAL 2007)*, (Lisbon, Portugal), 2007.
- [38] J. L. Arcos, R. L. De Mántaras, and X. Serra, “X. Serra, 1997. “SaxEx: a case-based reasoning system for generating expressive musical performances” ,” in *Proceedings of 1997 International Computer Music Conference* (P. Cook, ed.), (Thessalonikia, Greece), pp. 329--336, ICMA, San Francisco, 1997.
- [39] J. L. Arcos, R. L. De Mántaras, and X. Serra, “Saxex: A case-based reasoning system for generating expressive musical performances,” *Journal of New Music Research*, vol. 27, no. 3, pp. 194--210, 1998.
- [40] J. L. Arcos and R. L. De Mántaras, “An Interactive Case-Based Reasoning Approach for Generating Expressive Music,” *Journal of Applied Intelligence*, vol. 14, pp. 115-129, Jan. 2001.
- [41] T. Suzuki, T. Tokunaga, and H. Tanaka, “A case based approach to the generation of musical expression,” in *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, (Stockholm, Sweden), pp. 642--648, Morgan Kaufmann, San Francisco, 1999.
- [42] T. Suzuki, “Kagurame phase-II,” in *Proceedings of 2003 International Joint Conference on Artificial Intelligence (working Notes of RenCon Workshop)* (G. Gottlob and T. Walsh, eds.), (Acapulco, Mexico), Morgan Kaufmann, Los Altos, 2003.


- 
- [43] K. Hirata and R. Hiraga, “Ha-Hi-Hun: Performance rendering system of high controllability,” in *Proceedings of the ICAD 2002 Rencon Workshop on performance rendering systems*, (Kyoto, Japan), pp. 40--46, 2002.
- [44] G. Widmer, “Large-scale Induction of Expressive Performance Rules: First Quantitative Results,” in *Proceedings of the 2000 International Computer Music Conference* (I. Zannos, ed.), (Berlin, Germany), pp. 344--347, International Computer Music Association, San Francisco, 2000.
- [45] G. Widmer and A. Tobudic, “Machine discoveries: A few simple, robust local expression principles,” *Journal of New Music Research*, vol. 32, pp. 259--268, 2002.
- [46] G. Widmer, “Discovering simple rules in complex data: A meta-learning algorithm and some surprising musical discoveries,” *Artificial Intelligence*, vol. 146, pp. 129--148, 2003.
- [47] G. Widmer and A. Tobudic, “Playing Mozart by Analogy: Learning Multi-level Timing and Dynamics Strategies,” *Journal of New Music Research*, vol. 32, pp. 259--268, Sept. 2003.
- [48] A. Tobudic and G. Widmer, “Relational IBL in music with a new structural similarity measure,” in *Proceedings of the 13th International Conference on Inductive Logic Programming* (T. Horvath and A. Yamamoto, eds.), pp. 365--382, Springer Verlag, Berlin, 2003.
- [49] A. Tobudic and G. Widmer, “Learning to play Mozart: Recent improvements,” in *Proceedings of 2003 International Joint conference on Artificial Intelligence (Working Notes of IJCAI-03 Rencon Workshop)* (K. Hirata, ed.), (Acapulco, Mexico), 2003.
- [50] P. Dahlstedt, “Autonomous evolution of complete piano pieces and performances,” in *Proceedings of ECAL 2007 workshop on music and artificial life (Music AL 2007)*, (Lisbon, Portugal), 2007.
- [51] A. Kirke and E. Miranda, “Using a biophysically-constrained multi-agent system to combine expressive performance with algorithmic composition,” 2008.

- 
- [52] L. Carlson, A. Nordmark, and R. Wikilander, *Reason version 2.5 -- Getting Started*. Propellerhead Software, 2003.
- [53] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun, “Large Margin Methods for Structured and Interdependent Output Variables,” *Journal of Machine Learning Research*, vol. 6, pp. 1453--1484, 2005.
- [54] T. Joachims, T. Finley, and C.-N. J. Yu, “Cutting-plane training of structural SVMs,” *Machine Learning*, vol. 77, pp. 27--59, May 2009.
- [55] Y. Altun, I. Tsochantaridis, and T. Hofmann, “Hidden Markov Support Vector Machines,” in *Proceedings of the 20th International Conference on Machine Learning*, vol. 3, (Washington DC, USA), pp. 3--10, 2003.
- [56] M. Cuthbert and C. Ariza, “music21 [computer software],” 2013.
- [57] S. H. Lyu and S.-k. Jeng, “COMPUTER EXPRESSIVE MUSIC PERFORMANCE BY PHRASE-WISE MODELING,” in *workshop on Computer Music and Audio Technology*, 2012.
- [58] T. Joachims, “SVM^{hmm}: Sequence Tagging with Structural Support Vector Machines,” 2008.
- [59] R. P. Brent, *Algorithms for Minimization Without Derivatives*. 2013.
- [60] M. Hashida, T. Matsui, and H. Katayose, “A New Music Database Describing Deviation Information of Performance Expressions,” in *International Conference of Music Information Retrival (ISMIR)*, pp. 489--494, 2008.
- [61] S. Flossmann, W. Goebel, M. Grachten, B. Niedermayer, and G. Widmer, “The Magaloff project: An interim report,” *Journal of New Music Research*, vol. 39, no. 4, pp. 363--377, 2010.
- [62] W. Goebel, S. Flossmann, and G. Widmer, “Computational investigations into between-hand synchronization in piano playing: Magaloff’s complete Chopin,” in

Proceedings of the Sixth Sound and Music Computing Conference, pp. 291----296, 2009.



- [63] M. Grachten and G. Widmer, "Explaining musical expression as a mixture of basis functions," in *Proceedings of the 8th Sound and Music Computing Conference (SMC 2011)*, 2011.
- [64] S. Flossmann, W. Goebel, and G. Widmer, "Maintaining skill across the life span: Magaloff's entire Chopin at age 77," in *Proceedings of the International Symposium on Performance Science*, 2009.
- [65] M. Grachten and G. Widmer, "Linear basis models for prediction and analysis of musical expression," *Journal of New Music Research*, 2012.
- [66] S. Flossmann, M. Grachten, and G. Widmer, "Expressive performance rendering with probabilistic models," in *Guide to Computing for Expressive Music Performance* (A. Kirke and E. R. Miranda, eds.), pp. 75--98, Springer London, 2013.
- [67] S. Flossman and G. Widmer, "Toward a model of performance errors: A qualitative review of Magaloff's Chopin'," in *International Symposium on Performance Science*, (Utrecht), AEC, 2011.
- [68] S. Flossmann, W. Goebel, and G. Widmer, "The Magaloff corpus: An empirical error study," in *Proceedings of the 11th ICMPC*, (Seattle, Washington, USA), 2010.
- [69] G. Widmer, S. Flossmann, and M. Grachten, "YQX Plays Chopin," *AI Magazine*, vol. 30, p. 35, July 2009.
- [70] F. Lerdahl and R. S. Jackendoff, *A Generative Theory of Tonal Music*. 1983.
- [71] "KernScores." <http://kern.ccarh.org/>.
- [72] M. Clementi, *SONATINES pour Piano a 2 mains Op. 36 VOLUME I [Musical Score]*. Paris: Durand & Cie., plate d. & c. 9318 ed., 1915.

- 
- [73] M. Good, “MusicXML: An Internet-Friendly Format for Sheet Music,” in *XML Conference hosted by IDEAlliance*, 2001.
- [74] T. Joachims, T. Finley, and C. Yu, “Cutting-plane training of structural SVMs,” *Machine Learning*, vol. 77, pp. 27--59, May 2009.
- [75] E. Selfridge-Field, *Beyond MIDI: The Handbook of Musical Codes*. MIT Press, 1997.
- [76] “LilyPond.” <http://www.lilypond.org>.



Appendix A

Software Tools Used in This Research

REVIEW1 This research won't come into reality without many open-source software tools and free resources, we will walk you through a brief introduction to the softwares we used in this research. Please note that Internet resources come and go very quickly, if the links listed below are no longer valid, you can try to search it using search engines.

Linux Operating System

Most of the tools introduced below runs on modern Linux distributions. The distribution we are using is **Linux Mint Debian Edition (LMDE)**¹ (Linux kernel 3.10), which is a user-friendly Linux distribution based on Debian Testing. User who want to try music-related softwares without installing Linux on their harddrive can try **64 Studio**² Linux, which is a live CD distribution with many of the following softwares pre-installed. It also has many kernel optimization for real-time music manipulation. **Ubuntu Studio**³ is also an option, which has many pre-installed music softwares and is based on the popular Ubuntu Linux.

However, many Linux distribution uses PulseAudio audio server to manage audio device. But a badly configured PulseAudio server will introduce severe latency, which is not acceptable while doing MIDI recording. One workaround is to remove PulseAudio

¹http://www.linuxmint.com/download_lmde.php

²<http://www.64studio.com/>

³<http://ubuntustudio.org/>

and use raw ALSA (Advanced Linux Sound Architecture) driver instead. But be careful, hardware volume key may fail without PulseAudio.



Programming Languages

Python

Many researcher will choose Matlab or Octave for research projects because they have many useful toolboxes included. However, we believe that research project doesn't exist in vacuum. Drawing insight from the famous 80-20 rule, only 20% of the code are actually doing the core algorithm, the rest 80% are doing file manipulation, configuration, user interaction, and result display and plotting. Therefore, choosing a powerful and easy to write general-purpose programming language is extreme crucial. **Python**⁴ construct most of the infrastructure code for this project. Python is super easy to code, and has almost every tool you need to construct a fully functional experiment environment. We will highlight some useful module:

Music21⁵

We would like to give special thanks to the `music21` developemnt team. `Music21` is a python toolbox for music notation manipulation and analysis, developed by MIT. `Music21` can parse many score notations like MusicXML, MIDI⁶ and more into a very convenient `music21` object data-structure. Researcher can easily filter, split, search, and transform music notations. There are also many music analysis methods and feature extractors included. If you want to do music research, `music21` is a god-sent resource.

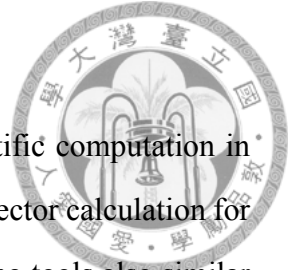
⁴<https://www.python.org/>

⁵<http://web.mit.edu/music21/>

⁶By default, `music21` will quantize MIDI input, so if you want to import MIDI recorded from human performance, you need to bypass the default parser and manually disable the quantizer

SciPy⁷

SciPy is a project that contains many useful toolboxes for scientific computation in Python. The SciPy core library and NumPy provides numerical and vector calculation for Python, with similar capability to Matlab. Matplotlib provides plotting tools also similar to Matlab. It's useful for small scale calculation. For heavy duty mathematical calculation, we suggest R programming language, which will be discussed in later section.



Simplejson

JSON (JavaScript Object Notation) is a plaintext data-interchange format, similar to XML but much light-weight. JSON is useful in experiment code for two purpose: first, JSON can serve as configuration file, it easy to parse and easy to edit. Second, JSON can serve as intermediate data file between each experiment module. For example, we use JSON to send extracted features from feature extractors to the machine learning module. Although plaintext takes more storage than binary file, but it's much easier for debugging because it's human readable. And you can simply parse the intermediate values and plot it using other plotting program. Python provides build-in support for JSON format via `json` and `simplejson` packages.

Argparse

Argparse provides command line argument parser for python scripts, using commandline arguments with configuration file, you can create very flexible, extendible and easy to automate scripts.

Logging

The built in `logging` module can print logging information with predefined format, and supports log level. By using log level, you can print debug information during development, and shutdown all debug message during production simply by changing the log

⁷<http://www.scipy.org/>

level flag.

R⁸



R is a programming language for statistical calculation, but it can also do general purpose math and plotting very well. R follows a functional programming design, so it may take some time to learn for people who only have experience in C/C++, Java or other imperative/Object-oriented programming language. But it is definitely a great tool for mathematical operations and plotting. We use R for experiment data analysis and for linear regression in early version of this research. R and Python can work seamlessly through the `rpy` package.

Score Manipulation and Corpora

MusicXML and MuseScore

MusicXML⁹ is a digital score notation format based on XML. It is well supported in most commercial music typesetting software. For other music typesetting format you may want to look at LilyPond. To view and edit musicXML score, we use the open-source software **MuseScore**¹⁰, it provides basic editing capability, and can export score as PDF. However, MuseScore often crash while loading bad-formatted musicXML file, so sometimes you need to look into its log file and fix the ill-formatted XML via a text editor.

Corpora

`Music21` contains a corpus¹¹, which will be automatically installed if you accept the licence term during `music21` installation. It covers a wide range of composers from early music, classical music to folk songs, with various genre and musical style. Another public

⁸<http://www.r-project.org/>

⁹<http://www.musicxml.com/>

¹⁰<http://musescore.org/>

¹¹<http://web.mit.edu/music21/doc/systemReference/referenceCorpus.html>

available corpus is called **KernScore**¹², which provides a better search engine. You can find works by composer, genre, form or other criteria. There are even a special section containing monophonic works. Scores from both corpus can be loaded and transformed in to desired format via `music21`.



MIDI Recording

Rosegarden¹³ is a digital audio workstation (DAW) software designed for MIDI. It can record, edit, mix and export MIDI tracks. To actually hear the music, you need a MIDI synthesizer to work with Rosegarden. **Timidity++**¹⁴ is built-in in many Linux distribution, and it provides a commandline interface to synthesize MIDI directly into a WAV file. However, the default sound quality from Timidity++ is not very satisfying, so we suggest using the qSynth, which is a QT front end for **FluidSynth**¹⁵. The default soundfont that comes with FluidSynth has very good sound quality.

With all these music softwares, it will soon be very hard to control the interconnection between software modules. This is when **JACK**¹⁶ comes to help. JACK is like a virtual “plug-board” for software that implements the JACK interface. It provides a central place in which you can control how the music data flow interconnects between software and hardware.

Audio Manipulation

When MIDI files are synthesized into WAV format, there are many tools that can help editing them. The most easy to use software with GUI is **Audacity**¹⁷, it can edit and mix audio tracks. For commandline tools (in case you need scripting), **lame**¹⁸ (MP3 encoder),

¹²<http://kern.ccarh.org/>

¹³<http://www.rosegardenmusic.com/>

¹⁴<http://timidity.sourceforge.net/>

¹⁵<http://sourceforge.net/projects/fluidsynth/>

¹⁶<http://jackaudio.org/>

¹⁷<http://audacity.sourceforge.net/>

¹⁸<http://lame.sourceforge.net/>

oggenc¹⁹(ogg vorbis encoder) and **FFmpeg**²⁰ are very helpful for file format transformation. To cut and combine audio tracks from commandline, use **SoX**²¹.



Data Visualization

As mentioned before, R and Matplotlib are good candidate for visualizing experiment data. But if you don't want to learn the syntax of R or Python, you can try **gnuplot**²². Gnuplot is a interactive (and scripting) environment for generating various types of plot like line plots or bar charts. It works particularly well if you use `grep` to extract datas for many files, say, extracting execution time information from logs.

SVM^{hmm}

SVM^{hmm}²³ is an implementation for Structural Support Vector Machine with Hidden Markov Model output. It's developed by Thorsten Joachims from Cornell University. It is based on SVM^{struct}, a more general framework for Structural Support Vector Machine. There are many other SVM^{struct} extensions such as Python or Matlab API.

Other

Sometimes the machine learning algorithm will run for a very long time. Then it's better if you can find a server that runs 24-7 in your home or laboratory. You can install a **ssh** server on that machine, and controls the experiment execution remotely. However, the experiment program will be terminated once you log out the `ssh` session. You can run your experiment program in **tmux**²⁴, a terminal multiplexer, instead. It will keep your program running even if you log out.

¹⁹<http://www.vorbis.com/>

²⁰<http://www.ffmpeg.org/>

²¹<http://sox.sourceforge.net/>

²²<http://www.gnuplot.info/>

²³http://www.cs.cornell.edu/people/tj/svm_light/svm_hmm.html

²⁴<http://tmux.sourceforge.net/>

Modern machines often have multi-core CPUs. But if your program only runs in one core, you waste the CPU resources and also your time. **Gnu-parallel**²⁵ can dispatch multiple instances of your script or program to each core. It will automatically find new job to run when the previous one is finished, so the CPU will always run on its full capacity.

We use **git**²⁶ for source control. \LaTeX ²⁷ is used to typeset this document.

Conclusion

We have reviewed many software tools used to construct this research. We want to emphasize that it is totally possible to use *only* free and open-source software to do all these heavy lifting. We encourage the reader to try these tools out, spread the words and even contribute to these projects. By doing so we can create a more friendly academia ecosystem and make the world a better place.

²⁵<http://www.gnu.org/software/parallel/>

²⁶<http://git-scm.com/>

²⁷<http://latex-project.org/>