

國立臺灣大學電機資訊學院電機工程學系

碩士論文

Department of Electrical Engineering

College of Electrical Engineering and Computer Science

National Taiwan University

Master Thesis



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

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○○○君（○學號○）在國立臺灣大學○○學系、所完成之碩（博）士學位論文，於民國○○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](#)

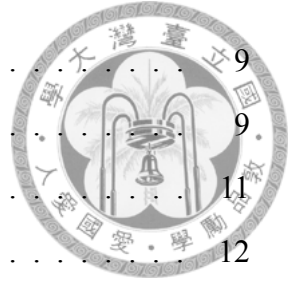
Key words:A, B, C, D, E, F, G



Contents

口試委員會審定書	i
致謝	ii
中文摘要	iii
Abstract	iv
Contents	v
List of Figures	viii
List of Tables	ix
1 Introduction	1
1.1 Motivation	1
1.2 Previous Works	1
1.3 Contribution	1
2 Previous Works	3
2.1 Goals	3
2.2 Methods	5
2.3 Special Features	7
3 Proposed Method	8
3.1 Overview	8

3.2	Features	9
3.2.1	Score Features	9
3.2.2	Performance Features	11
3.2.3	Normalizing Onset Timing	12
3.3	Learning Phase	13
3.4	Generating Phase	15
4	Structural Support Vector Machine	16
4.1	Background	16
4.2	Theoretical Background	17
4.3	Implementation	17
4.3.1	Quantization	18
5	Corpus	19
5.1	Score	19
5.2	Recording	20
5.3	Existing Corpora	21
5.4	Corpus Used	21
6	Experiments and Results	23
6.1	Onset Timing Normalization Selection	23
6.1.1	Experiment Design	23
6.1.2	Results	23
6.2	Parameter Selection	23
6.2.1	Experiment Design	23
6.2.2	Results	24
6.3	Turing Test	24
6.3.1	Experiment Design	24
6.3.2	Results	24



7 Conclusions

7.1 Future Works

A Software Tools for Music Research

Bibliography

25

25

26

27





List of Figures

3.1	System Architecture	9
3.2	Interval from/to neighbor notes	10
3.3	Duration from/to neighbor notes	11
3.4	Metric position	11



List of Tables



Chapter 1

Introduction

1.1 Motivation

Computer generated music, such as synthesized MIDI, are often considered robotic and unexpressive. But we have already witnessed the fluid and lively sound generated by state-of-the-art text-to-speech systems. This inspired us to develop a system that can read a music score and play it in an expressive, humanly way. Such system can be used for audiolizing score notation editing software, creating interactive media content, and generating royalty-free music.

Established pianists always has his/her own distinctive style. Such style distinguished himself/herself from all the other pianists. If the expressive performance system can learn the style of a performer, it might be able to provide musicological insight of performance styles. Furthermore, we can even make a master who is no longer with us play music he/she never played in his/her lifetime.

1.2 Previous Works

1.3 Contribution

The major contribution of this thesis is applying structural support vector machine on expressive performance problem. Also we tried to provide a public corpus for expressive

performance.





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 [1]. Early work includes the KTH system [1]. Reviewing computer expressive performance systems is a hard task, because there are exists large difference in the goals they wants to achieve, which makes evaluation and comparison difficult. To make things worse, although there is a contest called RenCon [2] in which researches compete their performances, there is no training and benchmarking corpus available, so if a system has never entered the contest and doesn't make its source code available, there is no way to compare them.

In this chapter, we will follow the categories summarized in [1]. Readers are suggest to refer to [1] because they gives very detailed discussion on all the systems. First we will discuss the various goals of computer expressive performance. Second, systems will be reviewed by their methods used. Finally, we will highlight some systems with special features.

2.1 Goals

The general goal of a computer expressive performance system is to generate expressive music, as opposed to the robotic and nu-human sound of rendered MIDI or other digital score format. But the definition of ``expressive" is ambiguous. The following are the most popular goals a computer expressive performance system wants to achieve:

1. Reproduce human performance.
2. Perform music notations in a non-robotic way.
3. Accompany a human performer.
4. Validate musicological models of expressive performance.
5. Directly render computer composed music works.



Systems that are designed to reproduce human performance usually has a certain performer in mind, like the Zenph re-performance CD [3] which will reproduce the performance style of Rachimaninov, it can even use Rachimaninov's style to perform pieces the musician never played in his lifetime.

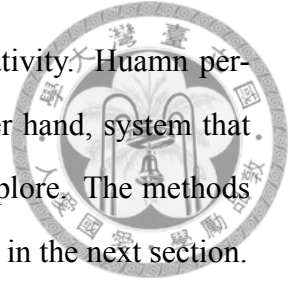
Some systems try to perform music notations in a non-robotic way, without a certain style in mind. These systems has been employed in music typesetting softwares, like Sibelius [4], to play the typesetted notation.

Accompaniment systems try to render expressive music that act as an accompaniment for human performer. This kind of systems has great value in music education. The challenge is that the system must be able to track the progress of a huamn performance, which is itself another established topic called score following. And the rendering must be done in real-time, which is uncommon in the field of computer expressive music performance.

The next goal is to validate musicological models. Musicologist may develop theories on music performance expressivity, and wants to validate them by experiments. These system focus more on the specific features that the theory tries to explain, so may not cover every aspect of performance.

Finally, some systems combines computer composition with performance. The benefit of this approach is that the performance module can understand the intention of the composition without the need to interpret them from the 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.

The goals discussed above imply different level of musical creativity. Human performance reproduction requires mimicry over creativity, on the other hand, system that plays its own composition can have a large range of creativity to explore. The methods employed also limits the ability of creativity, which will be discussed in the next section.



2.2 Methods

flute Despite the difference between goals of different expressive performance systems. All of them needs some way to create and apply performance knowledge on unexpressive 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 Hermodé Tuning System [6] focus more on intonation, thus it can generate expressions that requires string instruments techniques. Pop-E [?] is also a rule-based system which can generate polyphonic music, using its synchronization algorithm to synchronize voices. Computational Music Emotion Rule System [7] put more emphasis on rules that express human emotions. Other systems like Hierarchical Parabola System [5] [8] [9] [10], Composer Pulse System [11, 12], Bach Fugue System [?], Trumpet Synthesis System [?, ?] and Rubato [?, ?] 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 [?, ?, ?] and CaRo [?, ?, ?] both uses 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 [?] and Emotional flute [?] uses artificial neural network. ESP Piano [?] and Music Plus One [?, ?] uses Statistical Graphical Models, although the later one focus more on accompaniment task rather than rendering notation. KCCA Piano System [?] uses kernel regression. And Drumming System [?] tried different mapping models that generates drum patterns.

Evolutionary computation has also been applied, genetic programming is used in Genetic Programming Jazz Sax [?]. Other examples include the Sequential Covering Algorithm Genetic Algorithm [?], Generative Performance Genetic Algorithm [?] and Multi-Agent System with Imitation [?, ?]. 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.

Another approach is to use case-based reasoning to generate performance. SaxEx [?, ?, ?] use fuzzy rules based on emotions to generate Jazz saxophone performance. Kagurame [?, ?] style (Baroque, Romantic, Classic etc.) instead of emotion. Ha-Hi-Hun [?] 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 at el. called PLCG [?] uses data-mining to find rules for expressive performance. It's successor Phrase-decomposition/PLCG [?] added hierarchiacal phrase structures support to the original PLCG system. And the latest research in the series called DISTALL [?] 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 figures 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 compoisition. Ossia [?] and pMIMACS [?] 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.



2.3 Special Features

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 [?], trumpet [?], flute [?] and drums [?]. 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 [?], 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 [?] use a synchronization mechniasm to acheive polyphonic performance. Bach Fugue System [?] is created using the polyphonic rules in music theory about fugue, so it's inherently able to play polyphonic fugue. KCCA Piano System [?] can generate homophonic music -- an upper melody with an accompnaiment -- which is common in piano music. Music Plus One [?, ?, ?] is a little bit different because it's a accompaniment system, it adapts non-expressive orchastral accompaiment track to user's performance. Other systems usually generates monophonic tracks only.



Chapter 3

Proposed Method

3.1 Overview

The high-level architecture of the purposed system is shown in figure 3.1. The system is has two phases, the upper half of the figure is the learning phase, while the lower half is the generating phase. In the training phase, score and expressive performance recording pairs are feed in, their features will be extracted and used as the input for the learning algorithm. The learning algorithm will produce a learned model, which can be used to generate expressive performance hereafter. In the generating phase, a score is taken as input, and it's score features are extracted. The generation module ("Applying Model" box) then use the extracted features and the learned model to produce the preformance features for the score. The performance features can be viewed as the instruction for expression. With these features, an expressive MIDI performance for the input score can easily be generated.

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 dataset. The dataset is organized by tags, example of tags are like performer, mood, emotion, genre, style, etc. So the user can select a subset of training sample by selecting tags. Second, the phrasing 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

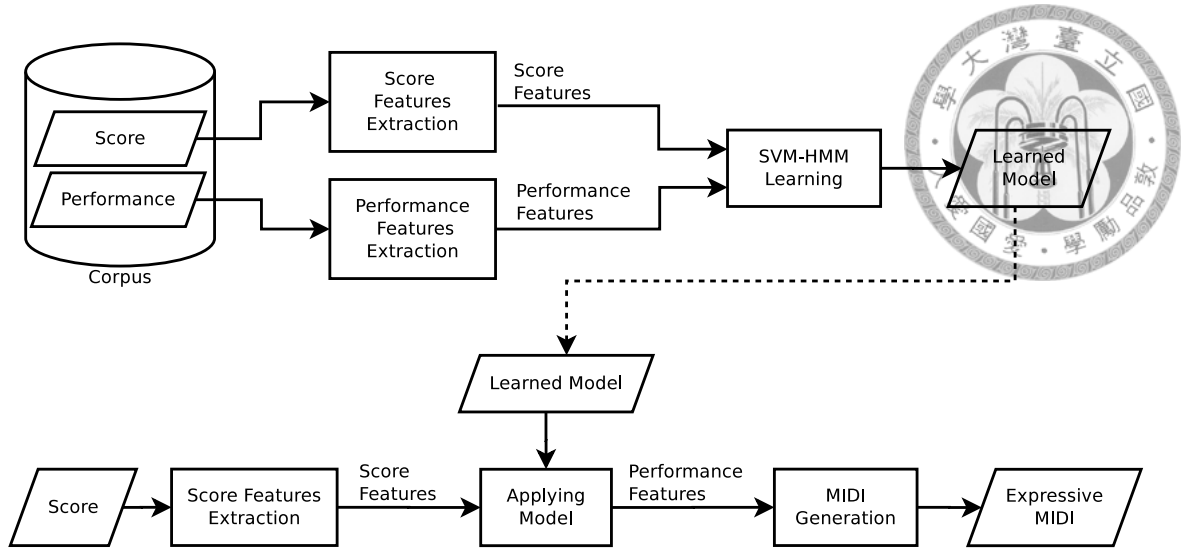


Figure 3.1: System Architecture

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 structural support vector machine, can only perform offline 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 instantly after loading the score. All the scores are loaded in batch, the system currently don't accept streaming input.

3.2 Features

The system is trying to mimic the process of human performance: the musician reads the explicit and implicit cues from the score and transform them into musical expressions. So the features can be categorized into two categories: score features and performance features. Score features are information contained in the score. Performance features corresponds to the musical expression. The basic time unit for both features are a note.

3.2.1 Score Features

Score features includes:

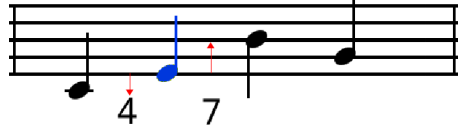


Figure 3.2: Interval from/to neighbor notes

Relative position in a phrase: The relative position of a note in the phrase. From 0% to 100%. This feature can catch the musical hint of the opening and closing of a phrase.

Relative pitch: The pitch (in semitone) of a note relative to the pitch range of the phrase.

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 direction of melody movement. Measured in semitone.

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

See figure 3.2 for example.

Interval to the next note: The direction of melody movement.

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

See figure 3.2 for example.

Note duration: The duration of a note in beats.

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 ,

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

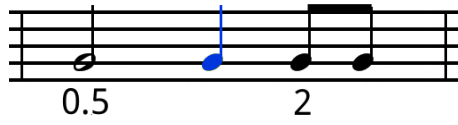


Figure 3.3: Duration from/to neighbor notes



Figure 3.4: Metric position

See figure 3.3 for example.

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

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

See figure 3.3 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. See figure 3.4.

3.2.2 Performance Features

Performance features includes:

Relative onset time bias: The onset time of a recording will not be exactly as the ones indicated on the score. Given a fixed tempo (beats/second), the score 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,

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

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.

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

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

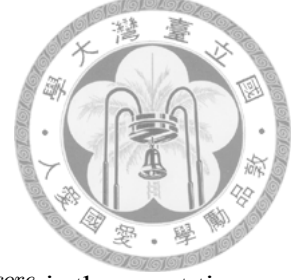
Relative duration: The actual duration of note divided by the total length of the phrase.

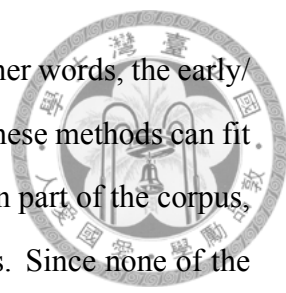
$$RD = \frac{D_i^{perf}}{length(phrase)}$$

3.2.3 Normalizing Onset Timing

The relative onset timing bias feature must be normalized to get meaningful result. For example, a phrase is played very fast by the performer, say, the played length is only 70% of the notation. If the first note is aligned, the onset timing bias will grow linearly until the end of the phrase, the last note will have a onset timing bias of about 30% of the phrase. But from the definition of this feature, the timing bias should be roughly less than half of a note's length, and it's mean should be around zero. This 30% value will introduce a relative large value in the training corpus, and thus there may be very large output value from the generation phase. If the model predict that a note should be played early or later for 30% of the phrase length, the melody will be messed up.

There are four possible type of normalization: the first note can either be aligned at onset, or not aligned at all. The last note can be aligned at onset, or aligned at note off event. The incentive for not aligning the first note is that the performer may intend to use an early start or delayed start as an expression, if the first note is aligned by it's onset, the first note in





every phrase will have a onset timing bias feature of value zero. In other words, the early/delayed start expression is lost. But experiments shows that none of these methods can fit all training samples. One may get good result by using one method on part of the corpus, but may result in large bias when applying it to the rest of the corpus. Since none of the methods can fit all, we take a different approach: let the program find the best fitting ratio for normalization. To achieve this goal, we first have to define how ``fit" two phrases are, or how to define the distance between phrase. 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 [?] is used to find the optimal ratio. To speed up the optimization and prevent extreme values, we imposed a range of $[initial_guess \times 0.5, initial_guess \times 2]$ to 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.

3.3 Learning Phase

In the learning phase, the features extracted in the previous stage is feed into a machine learning algorithm to produce a phrase model. The learning module has a input/output interface that is independent of the underlying algorithm, so different algorithm can be implemented without changing the overall structure of the system.

A training sample is loaded with the sample loader module, since a training sample is consisted of a score and a recording, the sample loader finds the two files given the sample name, and load them into `music21.Stream` object. 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

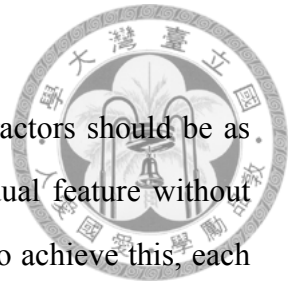
extractor module.

In order to keep the system architecture simple, the feature extractors should be as independent as possible, so the features can add or remove individual feature without breaking the system, and the features can be extracted in parallel. To achieve this, each feature must implement a common feature extractor interface, and feature extractors can't communicate with each other.

But sometimes a feature is based on other features, for example, the "relative duration with the previous note" is calculated based on the "duration" feature. But since they can't share informations, the "relative duration" feature extractor must call the "duration" feature extractor during its execution. To avoid redundant calling of feature extractors, we implemented a caching mechanism. Say, the "duration" feature had been extracted for a training sample, it's 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 re-calculation. This method can speed up the execution while keeping the system complexity low.

The the extracted features are aggregated into a json file, which will serve as the input for the learning algorithm. Each sample contains the extracted score and performance features. The learning algorithm can then do any pre-processing on the features, such as aggregation or quantization. The output of this module is the algorithm specific model description. For example, a linear regression algorithm will output the regression parameters. The algorithm is required to produce a model file containing the model description, but the system doesn't care about the internal format of the model description file, it will simply feed this model file to the generation module in the generation stage. So the developer of the learning module has to implement methods to write and read the model file themselves.

In the early stage of this research, linear regression is used. The results of linear regression is shown in [13]. In this thesis, Structural Support Vector Machine [14] is used instead. The detail of Structural SVM will be in the next Chapter.



3.4 Generating Phase

After the model for a input phrase is generated, we can than use the score features and model coefficients to calculated the performance parameters. These performance parameters will then be applied to the input score.

Some post-processing will be made for each performance parameters: The first time bias will be reduced if it is too negative and create a negative onset time for the first note. Loudness will be shift and shrink to a predefined range. The default is 80 127 MIDI loudness level. This will ensure the loudness in the output will be in acceptable range.

So after we input an input score, its score features will be extracted. These score features combined with regression model will be used to calculate performance features. The result will be an expressive MIDI output. Ready to be played by hardware or software synthesizer.





Chapter 4

Structural Support Vector Machine

4.1 Background

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 a 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.

4.2 Theoretical Background



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 $f : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}$, in which the input/output paris 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:

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

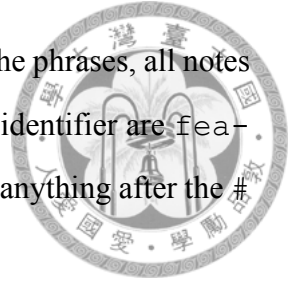
where \mathbf{w} is the parameter vector, and $\Phi(x, y)$ is the kernel function relating input x to output y . Φ can be defined to accomidate various kind of structures.

4.3 Implementation

Thorsten Joachims from Cornell University created a good toolbox for SVM-HMM learning called *SVM^{hmm}* [15]. According to the program's download page, the *SVM^{hmm}* is an implementation of structural SVMs for sequence tagging [?] using the training algorithm described in [?, ?] and [?]. The toolbox is contains two main program called `svm_hmm_learn` and `svm_hmm_classify`. The `svm_hmm_learn` takes a training file containing all the training sample. Each line in the training file is the featrues of a note, in the following format:

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

PERF is the 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 feature name : feature value pairs, separated by space. And anything after the `#` is considered as comments.



For architectural simplicity, one model is trained for each performance feature. The input for a model is all the score features, and the model will predict a single performance feature.

Because three performance features are used, three model file will be generated after running `svm_hmm_learn` on the

4.3.1 Quantization

One problem exist for using SVM-HMM on expressive performance: some of the features are continuous, but SVM-HMM can only generate discrete output label. Therefore quantization is required.



Chapter 5

Corpus

Since this research is based on a learning algorithm, a good expressive performance corpus is essential to its success. In this chapter, we will discuss what makes a good corpus and how to produce it.

A expressive performance corpus is a set of performance samples. Each sample consists of a score and a recording. The score is simply the music score being played, which contains notational information; and the recording is a digital or audio recording of a human musician playing the said score. With the two elements, a learning algorithm can learn how the music notation is translated into real performance. These two elements can come in many format, in the next sections we will discuss the pros and cons of some possible candidate.

5.1 Score

There are many way to represent a music score in machine-readable format, such as MusicXML [16], LilyPond [17], Finale, Sibelius, ABC, MuseData, and Humdrum. For more information, please check out [18]. For research purpose, proprietary format like Finale and Sibelius is abandoned because of their limited support from open source tools. MusicXML is based on XML (eXtensible Markup Language), it can express most music notations and metadata. LilyPond is a \LaTeX -like language for music typesetting. ABC, MuseData and Humdrum are based on ASCII codes and each defines their unique rep-

resentation for music score. Other formats such as image files (scanned or typeset by computer) or PDF files are an alternative, but they are not an option for direct computer analysis. MIDI can also be shown as music score in some music notation software, but MIDI is design as control signals for digital music equipments, so it lacks some music notations. Furthermore, the model of low-level music instrument control signals doesn't fit well with music notation, so it is not considered a good way to representation for music score.

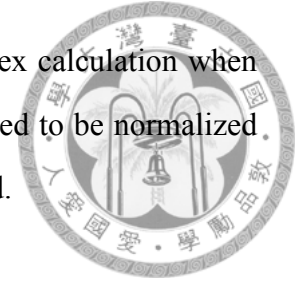
In this research, we use MusicXML as the main vehicle for music score, because of the following reasons: first, it covers most music notations and metadata need for this research. Second, it is supported in most music notation software, including the one used in this research -- MuseScore. Finally, the music21 toolbox can convert many other formats into MusicXML without problem.

5.2 Recording

A recording of expressive performance can be in MIDI or audio such as PCM WAV. Although audio recording has higher fidelity than MIDI, it takes extra effort to be analyzed by computer. Since onset detection and pitch detection algorithms still can't achieve perfect accuracy, manually labeling each notes required, which will soon become impossible to do when the texture of music become polyphonic. On the other hand, a multichannel MIDI recording can provide exact timing, key pressure, and pitch for each note, but it lacks timber and envelope information, which makes it hard to analyze how the musician use instrument-specific skills. Considering the above arguments, MIDI is used in this research.

A human musician can't play every note exactly on the beat, even if playing along with a metronome. There are two ways to record this behavior: first, record the exact note-on and note-off time, while keeping the tempo fixed; second, keep the notes on the beat, so the MIDI looks the same as the score. Then insert tempo-change event between each notes, so notes of the same length can be rendered differently because they have different tempo marker. The second method may look smart, because the score and performance can be

stored in one MIDI file instead of two, but it would involve complex calculation when linearly scaling the tempo. Since tempos from different samples need to be normalized during feature extraction, the first method is superior than the second.



5.3 Existing Corpora

5.4 Corpus Used

There are a few assumptions for the corpus used:

1. All the samples are monophonic. They only contain single line of melody, without chords.
2. A song is divided into phrases by manually labeling.
3. No human error exist in the recording; the score and recording are matched note-by-note.
4. The tempo label in MIDI recordings are the tempo by which the musician played.

For our corpus, we choose Clementi's Sonatina Op. 36. This is because Clementi's Sonatina is a basic repertoire almost every piano student in Asia will learn, so it's easy to find performers with different skill level to record the corpus. Clementi wrote these sonatinas in classical style, so the skill required to play them can be easily extended to other classical era works like Mozart or Haydn. This fact makes the learned model a very general one, which is good for evaluation. The digital score used is downloaded from KernScore website [19]. The original format is in Hundrum file format (.krn), then transformed into MusicXML by music21 toolkit. Because this research focus on monophonic melody only, so the chords in the scores are manually reduced to the highest note in the chord, which is usually the most salient melody line. All the other possible errors in the downloaded score is manually corrected to make them ready for printing and computer analysis. We use MuseScore notation editor to view and edit MusicXML; some metadata errors are corrected by editing the MusicXML with text editors.

To record the MIDI performances, we used a Yamaha MIDI keyboard with pressure sensitive keys. The Yamaha keyboard was connected to a MIDI-to-USB converter so it acted as a USB MIDI device on our Linux computer. On the Linux computer the Rosegarden Digital Audio Workstation (DAW) is used to record the MIDI. The Rosegarden DAW also generated the metronome sound to help the performer maintain a steady speed. One may argue that the tempo variation is also a part of the expression, but if the performer plays freely, the tempo is hard to determine afterwards, which will make learning scaling the performance in time domain very hard. The performers were allowed to record multiple times and edit the recording until there were no mistakes. The human error model is not part of this research now, so no mistakes are allowed in the performances to keep the problem simple.

After the MIDI files are recorded, some utility scripts we wrote are employed to make sure each recording is matched note-to-note with the corresponding score; if not, we will manually correct those mistakes. For example, if the pitch was played wrongly, we will correct the pitch but keep the onset, duration and intensity as is. The matched score and MIDI pair are then splat into phrases according to a phrasing 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) of the first note in a phrase in the score. The phrasing is assigned by the researcher now for accuracy, but any phrasing algorithm can be applied.

Because of the scope of the research, we didn't use all the information available, here are a few pieces of potentially useful information:

1. Polyphonic recording can provide information for polyphonic performance
2. The gaps between phrases can be learned to create models for inter-phrase delay.
3. Recordings with human error can make the system play more like human.



Chapter 6

Experiments and Results

6.1 Onset Timing Normalization Selection

6.1.1 Experiment Design

In section ??, four normalization method is proposed. To see which one is most robust, we validated them with empirical data. A robust normalization method should produce a mean-reverse sequence, not a monotonic increase or decrease sequence. Four types of normalization method is applied to every samples in the corpus, the result is shown in Figure ?? A regression analysis was applied to the result to see if there are clear trend in the normalized value.

6.1.2 Results

6.2 Parameter Selection

6.2.1 Experiment Design

Structural Support Vector Machine has some parameters that needed to be adjusted. We will leave the others to the defaults and change the SVM C trade-off parameter in this experiment. Since three models are learned for three performance features, we have three parameters to adjust.

[TODO: phrases count] phrases from [TODO:song counts] songs are used for training. Every first, fifth, and tenth phrases from each song is not included in the training sample, but used as testing samples. A three-by-three grid is layed out for three C parameters, each C takes the value of the powers of ten from [TODO: Cs] 10^{-5} to 10^4 , so [TODO: num of experiment] parameters are tested. Then the result is validated

1. Are all the output samples successfully generated? (Generation may fail if the performance features are unreasonable, for example, negative onset timing.)
2. Is the order of the notes preserved? Sometimes the first note is delayed too long and the second note is played too early, so the order is swaped.
3. Are there any extreme parameters that makes the expressive performance unnatural?

The first two criterias are checked by python scripts, the last one is done by manual inspection.

6.2.2 Results

6.3 Turing Test

6.3.1 Experiment Design

6.3.2 Results



Chapter 7

Conclusions

7.1 Future Works




Appendix A

Software Tools for Music Research



Bibliography

- [1] Alexis Kirke and Eduardo R. Miranda. An Overview of Computer Systems for Expressive Music Performance. In Alexis Kirke and Eduardo R. Miranda, editors, Guide to Computing for Expressive Music Performance, pages 1--47. Springer, 2013.
- [2] R Hiraga, R Bresin, K Hirata, and RenCon KH. Turing test for musical expression proceedings of international conference on new interfaces for musical expression. In Y Nagashima and M Lyons, editors, Proceedings of 2004 new interfaces for musical expression conference, pages 120--123, Hamatsu, Japan, 2004. ACM Press.
- [3] Rachmianinoff - Plays Rachmaninoff. <https://www.zenph.com/rachmaninoff-plays-rachmaninoff>, 2009.
- [4] Sibelius. <http://www.avid.com/us/products/sibelius/pc/Play-perform-and-share>.
- [5] Anders Friberg, Roberto Bresin, and Johan Sundberg. Overview of the KTH rule system for musical performance. Advances in Cognitive Psychology, 2(2):145--161, January 2006.
- [6] William A. Sethares. Tuning, Timbre, Spectrum, Scale. Springer, 2005.
- [7] Steven R. Livingstone, Ralf Mühlberger, Andrew R. Brown, and Andrew Loch. Controlling musical emotionality: an affective computational architecture for influencing musical emotions. Digital Creativity, 18(1):43--53, March 2007.
- [8] Neil Todd. A computational model of rubato. Contemporary Music Review, 3(1): 69--88, January 1989.

- 
- [9] Neil P. McAngus Todd. The dynamics of dynamics: A model of musical expression. The Journal of the Acoustical Society of America, 91(6):3540, June 1992.
- [10] Neil P. McAngus Todd. The kinematics of musical expression. The Journal of the Acoustical Society of America, 97(3):1940, March 1995.
- [11] M Clynes. Generative principles of musical thought: Integration of microstructure with structure. Journal For The Integrated Study Of Artificial ..., 1986.
- [12] M Clynes. Microstructural musical linguistics: composers' pulses are liked most by the best musicians. Cognition, 1995.
- [13] Shing Hermes Lyu and Shyh-kang Jeng. COMPUTER EXPRESSIVE MUSIC PERFORMANCE BY PHRASE-WISE MODELING. In workshop on Computer Music and Audio Technology, 2012.
- [14] Thorsten Joachims, Thomas Finley, and CNJ Yu. Cutting-plane training of structural SVMs. Machine Learning, 77(1):27--59, May 2009.
- [15] Thorsten Joachims. SVM^{hmm}: Sequence Tagging with Structural Support Vector Machines, 2008.
- [16] Michael Good. MusicXML: An Internet-Friendly Format for Sheet Music. In XML Conference hosted by IDEAlliance, 2001.
- [17] LilyPond. <http://www.lilypond.org>.
- [18] Eleanor Selfridge-Field. Beyond MIDI: The Handbook of Musical Codes. MIT Press, 1997.
- [19] KernScores. <http://kern.ccarh.org/>.