# RE-TARGETING EXPRESSIVE MUSICAL STYLE USING A MACHINE LEARNING METHOD

*Simon Lui*

Department of Computer Science and Engineering, Hong Kong Univeristy of Science and Technology, Hong Kong
virtuoso@cse.ust.hk

*Andrew Horner*

Department of Computer Science and Engineering, Hong Kong Univeristy of Science and Technology, Hong Kong
horner@cse.ust.hk

**ABSTRACT**

Expressive musical performing style involves more than what is simply represented on the score. Performers imprint their personal style on each performances based on their musical understanding. Expressive musical performing style makes the music come alive by shaping the music through continuous variation. It is observed that the musical style can be represented by appropriate numerical parameters, where most parameters are related to the dynamics. It is also observed that performers tends to perform music sections and motives of similar shape in similar ways, where music sections and motives can be identified by an automatic phrasing algorithm. An experiment is proposed for producing expressive music from raw quantized music files using machine-learning methods like Support Vector Machines. Experimental results show that it is possible to induce some of a performer's style by using the music parameters extracted from the audio recordings of their real performance.

## 1. INTRODUCTION

It has been a hot topic recently to develop computational methods for expressive music performance. Expressive musical performance is more than just a simple variation in tempo and dynamic. The performing artist is an indispensable part of the music, deriving information from their understanding and musical knowledge. Not every expressive performance feature can be represented in music notation – something composers are well aware of. Hence, to understand expressive performance, we must study the musical behaviour of performers. Typically, researchers have built formal models of expressive performance based on real musical performance. In order to build models with strong empirical foundations, inductive methods must be introduced, such that a large amount of real-world performance data is used as the basis for the model. To deal with the complexity of such a large amount of data, we make use of machine learning and data mining.

There are a large variety of musical descriptors that can be investigated. These descriptors range from low-level features, such as RMS envelope and spectral shape, to high-level descriptors such as terms like "delightful" and "sad" music. High-level terms can also be described by a combination of low-level audio descriptors. A common question of interest is, whether it is possible to represent expressive styles in terms of these descriptors in a digital format. Previous research has shown that music styles can be represented, to certain extent, as the deviation of three fundamental parameters: dynamics, tempo and articulation [1]. Previous research suggests that different musicians usually perform the same piece in a similar way in aspects like dynamics, due to music context, structure, common musical sense, and so on. However there are also slight differences between different musicians [2]. Each musician has a unique performing style, where some particular performing features will uniquely and frequently appear in different pieces played by the same performer [3].

Experimental results also show that by collecting several pieces performed by the same musician, it is possible to train a set of performance style parameters from the performance data, where the trained data can be used to distinguish the performance style of that particular performer from others [4]. Successful learning from even extremely limited training data can still be achieved by making use of ensemble learning. Once learned, the extracted "performance style" can be applied to a raw note list to make it expressive.

We propose an experiment where musical style is induced by multiple Support Vector Machines and applied to MIDI note lists in order to produce expressive musical performances. We first describe some musical facts observed from several expressive performance excerpts. We then discuss the implementation of our proposed experiment. Finally we conclude with the listening and statistical test results, as well as our future plans.

## 2. OBSERVATIONS

Our program has been developed on the basis of the following observations, which have been carefully tested, supported with strong reasons, and prolific examples. This is an essential step in our program development.

### 2.1. Global dynamic trend

Figure 1 shows a smoothed dynamic graph and a smoothed pitch graph of the Sonata No.1 in G minor BWV 1001, second movement (Fuga), by J.S. Bach, performed by Jascha Heifetz. The complete piece of music is smoothed by a sliding window of 8 bars (32 beats). The trends of the two graphs are very similar. This is not an isolated case. We sampled 6 different performers as well as different movements of the Bach Sonata and Partita, finding that they all return similar trends for the two graphs. Four of the results are shown in Figure 10. The global dynamic trend closely follows the global pitch trend. In most cases, the higher the pitch, the higher the dynamics. The only difference between different performers and different music is the trend ratio.

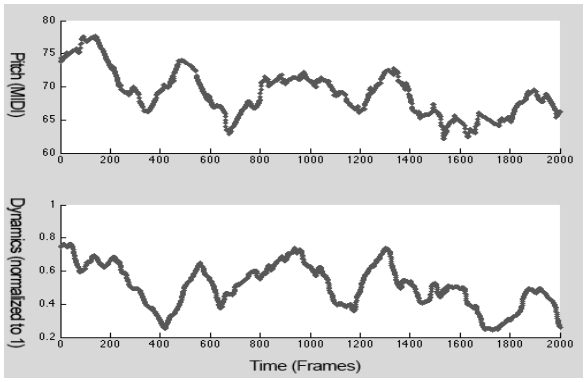
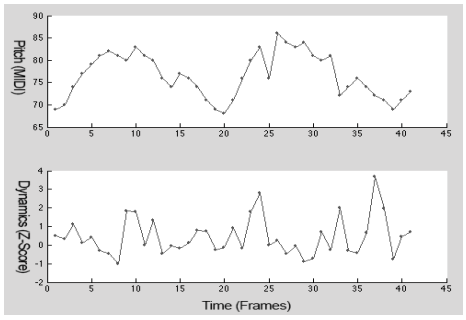


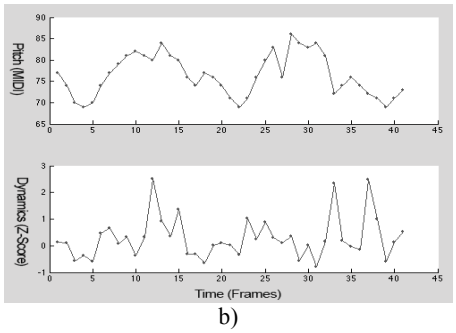

Figure 1. *A smoothed graph from the Sonata No.1 in G minor BWV 1001, second movement (Fuga) by J.S. Bach, performed by Jascha Heifetz. Top: The smoothed pitch graph. Bottom: The smoothed dynamic graph.*

### 2.2. Local dynamic change

We performed careful observations on every position for several movements of the Bach Sonata and Partita. We first magnified a small portion of two bars into a full screen on our computer. We compared local dynamic changes with corresponding local pitch changes, using several preprocessing methods including logarithmic, smoothing, standard score, deviation chart and so on. We eventually found that the standard score (Z-score) reflects the relationship between local dynamics and pitch changes well. Figure 2 shows two examples. The pitch trend in (a) is similar to (b), and the trend of Z-score of their relative dynamic levels looks very similar.




a)

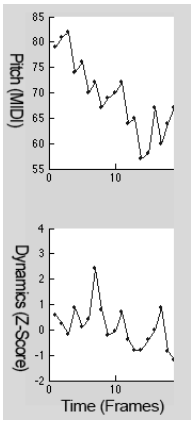



b)

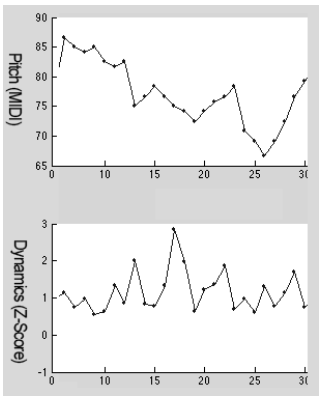
Figure 2. *Two excerpts from the Partita No.2 in D minor, BWV 1004 4th movement (Giga) performed by Itzhak Perlman.*

Next, we found that phrases with similar pitch patterns but different scales also had similar dynamic patterns in their own scales as shown in Figure 3.

To conclude, we believe it is possible to describe a performer's dynamic style by combining his/her global and local dynamic trends. As a violinist, this assumption well-matches my behaviour as a performer: imagine when we first look at a piece of music, we initially picture the whole piece from a global point of view, planning for the roles of different sections. However for each small motive we customize it to a personal performance style. We are very likely to perform with similar dynamic patterns for phrases with similar fingering patterns, hence it is reasonable for similar local pitch trends to have similar local dynamic trends.




a)




b)

Figure 3. *Two excerpts from the Partita No.2 in D minor, BWV 1004 4th movement (Giga), performed by Itzhak Perlman.*

### 2.3. Feature vectors

It is possible to use a few fundamental features to fully describe a performance style. Gerhard Widmer showed that dynamics, tempo, and articulation are adequate in representing the performance style of a piano solo. He successfully classified performance style of different pianists by using dynamic and tempo relationships.

From our point of view, dynamics and articulations are essential features for describing an expressive performance style. Moreover, for vocal, wind, brass, bowed string instruments and vocal, performing techniques like vibrato and glissando are also essential features for describing their performing style. We describe these features as pitch bend. Tempo is also essential for piano solos and perhaps for most other solo performances such as harp and guitar. However, much other type of music is performed in ensemble form. These are as important as solo performances, and probably there are more ensemble recordings than solo recordings in existence. If the ensemble music is classical music, the performers usually have to follow the tempo of the conductor or the quartet leader; for pop music, the lead instrument or singer usually has to follow a metronome or the tempo of the drummer, as most drummers are actually following a metronome beat through a headset. We measured the tempo of a number of pop pieces, symphonies and solo concertos, using the beat tracking program by Dixon [5]. Dixon's algorithm ranks the 1st in the Audio Beat Tracking task in Music Information Retrieval Evaluation eXchange (MIREX) 2006, which is very accurate and efficient. We found that the global tempo was steady most of the time. So, we believe global tempo is a good style indicator for solo performances, but not for music performed by more than one player. Although there is still a little local deviation in tempo, but the deviation is limited within a small range of the global tempo, where the performer cannot go ahead too much and he has to return to the original tempo in a bar or two. Since the global tempo is almost steady, we interpret this tempo deviation as lengthening and shortening of notes, which is the scope of articulation.

It is observed that the dynamic feature can describe the articulation feature. Articulation refers to the length of music note and it can be described by its note-end position: it is a note-end when the dynamic value drops below certain threshold. To conclude, we believe that dynamics, sometimes together with pitch bend, can fully represent music performance style. In this paper, we first focus on dynamics, since it applies to all musical instruments.

## 3. EXPERIMENTAL SETTING

### 3.1. Music data

Expressive performance recordings were extracted from an audio CD. Wave audio files in simple PCM format (44 kHz, 16 bit) were used. The input music data were assumed to be in MIDI format 1, following the GM (General MIDI) standard in order to standardize the velocity and channel parameters.

We chose the unaccompanied Sonata and Partita for the violin solo BWV1001-1006 by J.S. Bach, since it is a large set of accompanied works that can provide many clean samples for training. It is also one of the most famous violin masterpieces, hence it is easy to find many different versions by different performers. The reason for choosing Bach's music is because his music has dense and conscientious musical structures, for example, fugue and counterpoint. We believe that it is relatively easy to find re-occurring patterns in Bach's music, hence it should be a good starting point. For each piece, we prepared certain versions performed by different famous violinists including Itzhak Perlman, Jascha Heifetz, Midori Goto and Gil Shaham. The author, Simon Lui, also included performances for some excerpts as well, since we can record the same piece for an unlimited number of times, which can help us find the differences and similarity of a sample piece performed by the same performer. The MIDI data were downloaded from the Classical MIDI archive. All MIDI files were quantized and tidied for ready use. We used the Vienna Symphonic Library Strings Pro edition as an output sound sampler, and we used Logic Pro 8 as a sequencer to process the string instrument library.

### 3.2. Support Vector Machine

For the SVM machine, our experiments used LIBSVM Version 2.88 developed by the National Taiwan University, which was implemented by Chih-Chung Chang and Chih-Jen Lin [6].
Since SVMs require each data instance to be represented in numerical format, we used the GM (General MIDI) digital value to represent the vector value. Each support vector contained 64 frames, and each data represented the pitch value (1-127, in the GM standard) at a certain time.

In this support vector design, both rhythmic and pitch changes were included. 64-frame-level data was considered since most pieces rarely have notes durations shorter then a 128th notes. One support vector represents the features within 8 beats (i.e., 2 bars), so 8 / (1/128) = 64-frame-level data is considered. If the 256th note is present, then probably an 8th note instead of a 4th note could be tracked as a beat, so 128th notes could be identified by the program relatively as a 64th notes. Also, the inexpressive MIDI score files used were quantized and non-expressive, so there were no grace notes or acciaccatura. Hence the support vector of the 64-dimensions has fine enough resolution.

## 4. IMPLEMENTATION

### 4.1. Note identification

First, we identify musical notes from an audio file by pitch extraction. We used a modified pitch extraction algorithm suggested by Peeters [7] which is a very fast and accurate. Assume the fundamental frequency of a harmonic tone is f. The FFT of this note should peak at f and its multiples. On the other hand, the cepstrum of this note should peaks at f and its divisors, because cepstrum shows the repeat rate of the peaks and crests in the corresponding FFT graph, while the highest repeat rate of peaks in the FFT graph are probably the fundamental frequency and its divisors.

Hence to conclude, the cross product of a FFT and cepstrum should produce a graph that peaks at the fundamental frequency. Peeters suggested that the cross product of the autocorrelation of a Discrete Fourier Transform and cepstrum can reach the highest accuracy of 97%, while the cross product of a FFT and cepstrum has the second highest accuracy of 91.4%.

We further increase the accuracy of the pitch estimation by a low-pass filter of 20Hz and doing a natural logarithm in order to sharpen the peak of the FFT and cepstrum. Moreover, since we extract pitch data for every frame, and each note actually lasts for at least 8 frames, an error correction technique can be applied to fix almost all the errors: firstly, discard discontinuous frames with lengths less than a 32nd note; secondly, for gaps shorter than a 32nd note, are filled in with the next/previous pitch.

We tested the accuracy for an excerpt from the Partita No.2 in D minor, BWV 1004 4th movement (Giga) performed by

Itzhak Perlman. The final pitch estimation was improved to 96% with this simple error correction technique. The remaining 4% error was mostly due to the performer's slightly off-key performance. We carefully looked into some miscalculated cases, and all of them were actually off-key. In conclusion, there is almost no room for further improvement to the pitch estimation accuracy for monophonic audio.

Sometimes we still got a dirty cepstrum graph even though the music was monophonic. This was mainly the reverb from the previous note. This error was solved by sharpening the graph by a prior natural logarithm. After obtaining the fundamental frequency for each note, we converted them to the nearest pitches, and then to the General MIDI parameters.

### 4.2. Beat Tracking

The audio file data was then compared with the MIDI file. However, before doing the comparison, the audio file data has to be mapped to the corresponding notes in the MIDI file. To do this, the beat position of each note has to be calculated by beat tracking.

First, we calculate the IOI (inter onset interval) for all note-on times, and then perform clustering for the IOI values. IOIs of difference less than 25% were joined together. When all the IOIs were merged, clusters were merged with a 25% threshold. Finally, the cluster with the most number of members was taken as the beat of the piece.

Beat tracking was performed for both audio data and source MIDI. The beat tracking result was not 100% accurate, sometimes it over- or under-estimated the beat by a scale of two or one half. However it did not affect the accuracy of our machine, since we are only looking for reference points in the same piece of music, while the reference points between MIDI and audio data always match perfectly.

Since we only target the largest final cluster, it is a waste of time to merge clusters after we processed each IOI. Instead, we can merge the clusters once only after all the IOI are merged, and the result is exactly the same as our last implementation.

### 4.3. Note Mapping

After beat tracking, we locate a set of reference points in the MIDI and audio files, hence we can perform note mapping between the audio and MIDI data. We map every MIDI note to its corresponding audio note instead of the reverse. This is because the pitch data in the audio files, which are extracted by the pitch extraction algorithm, might not be 100% correct even though the error rate is small. On the other hand, the MIDI file is a source reference file and hence its pitch data is 100% correct. Hence we should do the mapping from the MIDI notes to audio notes.

We use a 96-frame window for mapping notes in a 64-frame vector, which is 150% of the target size. We search for the correct mapping pattern in a binary string format. The binary string with the largest sum of values is the best-matched sequence. The pseudocode of the note-mapping algorithm is as follows:

```
for trial = 1 to 2^windowSize-1,
    for bit = 1 to windowSize,
            bitString(windowSize-bit+1)
            = floor(mod(trial/(2^(bit-1)),2));
    end
    match midiString with bitString,
    producing matchedString composed of 1 and 0.
    count how many 1 are there in matchedSting,
    break if at least 90% match.
end
```

Finally we prepare data in the SVM vector format. We use the pitch as a vector feature, and the dynamic cluster code as the class. Since the note mapping results are mostly 100% full mapping, it is more efficient to try from an all '1' sequence in order to save computation time.

### 4.4. Segmentation

After note mapping is done, we segment the music into phrases. At an early stage of development, no segmentation was done, and we considered each note by a sliding window with a fixed size of 8 beats. However the learned performing style was not accurate. This implementation did not match how a performer thinks: a performer makes judgment about their own performing styles by motif, not by phrases of fixed length.

Hence in the second stage, we processed the input phrase-by-phrase, where the phrase segmentation was done manually. The whole piece was first divided into a few (four to eight) musical forms, and then within each form similar musical phrases were identified and motives of a half-bar to 4-bars were identified.

However, in order to build up an accurate SVM machine, we needed a large amount of segmented music samples. It was too slow to do the segmentation manually and we needed an automatic solution. Hence a modified version of the Phrase Stealing Algorithm by Lui [8] was used to perform auto segmentation. The originally Phrase Stealing Algorithm identifies music phrases by tying individual music notes together according to a voice leading table. In this experiment, we tie music notes together according to harmony progression. First, we segmented the music by long notes and rests, resulting in sets of music chunks. Music chunks shorter than 64 frames could be readily used as phrases. We did further segmentation for those music chunks which were longer than 64 frames. Within each chunk, for each note, a list of expected chord was calculated, resulting in an 2D "expected chord matrix". The elements in the expected chord matrix were tied according to a self-made "chord progression table". Finally all the tied phrases were viewed as motives.

Motives of different lengths were normalized to a fixed length of 64-frames. This process is based on the observation quoted in section 5.3.2 that similar pitch trends of different scales also have similar dynamic trends. As a result, we get a set of note vectors, all of length 64-frames.

The FFT process is actually the most serious bottleneck of the whole program. However, the FFT process for different portions of music is independent and can actually be done in parallel. The FFT process can be speeded up by dividing the piece of music into several chunks, and performing FFT calculations simultaneously in different threads. Originally, we used Matlab 2006a which is a single thread application. However, we can still perform multitasking with limited power by making use of Basic Linear Algebra Subroutines. We have to do these environment variable settings outside Matlab:

| | |
|---|---|
| *BLAS_VERSION* | *mkl.dll* |
| *OMP_NUM_THREADS* | *(number of threads)* |

Finally, we switched to Matlab 2008a which supports multi-threading. Setting can easily be done within the Matlab code, and the performance was greatly improved.

### 4.5. Absolute dynamic data

To calculate absolute dynamic levels, the root mean square (RMS) value of the signal amplitude was taken and only the RMS peak of each frame was used. The dynamic value of the whole piece of music was generalized to the MIDI scale of a range of 0-127, where the quietest note in the whole piece is indicated as volume 0 and the loudest note is indicated as volume 127. We choose a relative dynamic measure rather than an absolute measure because of human perception. Most people cannot tell if a certain single tone is loud or quiet, but everyone finds a 60db voice louder than a 30db voice. Hence we do not describe the dynamics in absolute values such as p, mp, mf and f.

In order to represent the articulation information in the dynamic vector, we set the dynamic threshold of defining music note-ends. We first measure the ratio of silent period within the whole MIDI score. Then, we plot an accumulative histogram of the dynamic value of the whole audio recording. The dynamic value at the index of the silent-period-ratio is the dynamic threshold of defining note-end. For audio frame with dynamic value below this threshold, the dynamic value is set to be 0.

### 4.6. Global dynamic ratio

To calculate the global dynamic ratio, the whole list of dynamics and pitch was first smoothed by a window of 8 bars, which is around the size of two to four motives. The global trend ratio is in a linear form as follows:

$$\text{GlobalTrendRatio} = \frac{1}{\text{numOfSample}} \sum \ln(\text{dynamic}) / \ln(\text{pitch}) \quad (1)$$

This global trend ratio can represent the global dynamic of each musical section.

### 4.7. Local dynamic vector

To calculate the local dynamic vector, the RMS value of the dynamic data was not used directly, but we further reduced the global factor by using a standard score (Z-Score). The Z-Score calculates the local change of a note compared with the local mean, regardless of the standard deviation of the population. The Z-score can be calculated as follows:

$$Z = \frac{x - \mu}{\sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}} \quad (2)$$

Each dynamic vector is smoothed by a window of one beat. We can borrow notes from the next / previous window when smoothing the beginning and end of each vector. For the beginning and end of the piece, we simply decrease the length of the window.

The smoothing process helps in representing the general trend of the relative dynamic change within a vector. It actually sounds more natural to express the general intention of the performer rather than reproducing each digit from the data source. To clarify this, we played a short excerpt from the same movement of the Bach Partita three times, all with the same dynamic intention of a crescendo and then diminuendo. The three dynamic graphs look a bit different but the smoothed versions looks almost the same. Here we conclude that with the same expressive intention, the resulting original data can be different, while the smoothed data will look very similar. Further, we seldom find more than 2 global peaks in each dynamic vector. Perhaps it is possible to generalize the vector by a formula. We will try this at the next stage of development.

Next, all elements in each pitch and dynamic vector are subtracted by the value of the first element in the corresponding vector. Hence, the vector represents the relative pitch and dynamic change compared with the first note.

After smoothing and scaling, we perform clustering on the set of dynamic vectors. Each vector joins a cluster if the difference between the cluster value and the squared sum of its feature components is the minimum among all clusters and is less than 300. We choose 300 as a threshold based on the observation that the z-scores mostly range from -4 to +4. The difference between the absolute crescendo and absolute diminuendo vector is 1430. In this case, the two vectors should never be in the same cluster:

$$2\sum_{i=1}^{32}\left(\frac{4+4}{64/2}i\right)^2 = 1430 \quad (3)$$

The difference between two almost parallel vectors which have linear average values of +1 and -1 is around 2 x 2 x 64 = 256. In this case, the two vectors should be in the same cluster, while this should be the upper bound threshold for a vector to join a certain cluster. Since we aim at re-targeting individual and expressive performing styles, it is fine to over-fit the clustering process since the style can still be preserved in different clusters. However, we have to avoid loose clustering which alter the shape of the performing styles too much. Hence a threshold of 300 will be 4 times away from 1430 and just fit the squared sum difference of 256. Clustering values can be fine tuned in the future, which only alters the number of clusters produced and the precision of the style data.

Instead of the brute force cluster merging approach, we speeded up the merging process by only considering clusters that have just been changed. However, we cannot leave all the cluster merging work to the end as we did in beat tracking (see session 6.2.3), since we need to assign the cluster number to each dynamic vector after it is clustered. Hence we need to update the cluster list for every vector calculation run.

### 4.8. SVM training and prediction

We use SVM for training because we do not want to over-fit the data. In a real world example, there are always inseparable feature vectors. An over-computed separation formula will waste a lot of computation time. The trade-off between using a less complicated method is perhaps a few incorrectly classified points. Actually it is more practical and accurate to give up a few scattered feature vectors.

We estimate the SVM Kernel parameters by using a systematical optimal model parameter search. The best way is to start with n-fold cross validation. We first divide the training set into n subsets of equal size, sequentially one subset is tested us-

ing the classifier trained on the remaining n-1 subsets. The most suitable parameters should give the best results in cross validation tests.

We chose the radial basis function (RBF) kernel [9] among the four existing kernels (linear, polynomial, RBF, and sigmoid). The linear kernel is too rough and should not be used. The polynomial kernel has too many hyper-parameters, which increases the complexity and hence the running time of the parameter search. The sigmoid kernel behaves like the RBF kernel in many cases, but with no real advantages over it. For some parameters the sigmoid kernel is not valid. To avoid unnecessary failure of the program, we choose RBF kernel which is proved to be the best kernel of all for our purposes.

Originally, we developed a graphical version of LIBSVM, where the user only needed very little input, and the remaining textboxes were automatically filled with suggested parameters. However, we finally achieved an automatic estimation of all parameters so we revised the skeleton and embed the code into the Matlab structure as a one-click design.

### 4.9. Re-targeting music

Lastly, the selected expressive performance style is re-targeted to a raw note list of MIDI. The global dynamic data is first calculated with the global trend ratio of the selected performer. Then the local dynamic data is predicted with the SVM machine. The local dynamic data is then merged with the global dynamic data in order to produce the actual dynamic level of the performance. The resulting dynamic data is then converted to the MIDI GM1 data format which produces an expressive MIDI performance file. The expressive MIDI file is finally rendered with a software sampler to produce an expressive audio performance file. An overview of the whole process is shown in Appendix I.

## 5. TESTS

### 5.1. Listening test setting

First of all, we decided not to compare the machine's output with the original wave file. Comparison is fair if and only if it is performed under the same environment. It would be unfair if the files to be compared were produced from different sound sources. Hence by comparing audio file output from the same source, the following tests reflect the performance of our machine rather than the quality of the original audio or the sound sampler.

Twelve listeners were invited to do the test. Eight of them were musically trained while seven of them could play the violin and knew the Bach Partita very well. Four of them were not musically trained but all of them enjoyed listening to music. The listeners were required to sit in a quiet room using headphones. After the following sequences were played: o*riginal file, predicted output file, original file,* they had to rate on the performance of *predicted output file:* 7 being the best, and 1 being the worst, as shown in Table 1.

All clips were music excerpts of 6 seconds, each listener performed 4 sessions for each of the three tests. The tests were very short since the tests were intensive in nature where fatigues highly affect the accuracy of the test.

#### *5.1.1. Test 1: basic accuracy*

First, the expressive performance data of an audio file was extracted. The expressive data was directly applied to its MIDI source file, producing an expressive *MIDI file A*. Then the expressive data was used to build an SVM machine. Using the SVM machine, performance parameters are predicted using the MIDI source file as input, producing an expressive *MIDI file B*. Both *MIDI files A and B* were passed through the sound module to produces two audio files, then the listeners judged their performances.

#### *5.1.2. Test 2: influence of extra feature vectors*

Similar to test 1, the expressive performances from four audio files were extracted. The SVM machine was built with these four different pieces of music, producing four pairs of audio files. Listeners judged the performance between different pairs of output files.

#### *5.1.3. Test 3: ability of predicting unseen data*

Similar to test 2, the expressive performances from four audio files were extracted. However, the SVM machine was built with three of them only, in order to predict the performance of the remaining unseen piece of music. Listener judged the performance between different pairs of audio output files.

### 5.2. Listening test result

The test results are shown in Table 1. Listeners found all the music very natural in tests 1a, 2a and 3a, because the style data were extracted from real performances, and the design of describing dynamics as a combination of global and local portion was successful.

For classification between styles, both tests 1b and 2b obtained good results, while test 3b was just fine, because test 1b and 2b include the original wave file in the training set. The result of 3b could be improved after the implementation of articulation learning.

Table 1. The *listening test result.*

| **Test** | 1a | 1b | 2a | 2b | 3a | 3b |
|---|---|---|---|---|---|---|
| **Rate** | 6.52 | 6.21 | 6.67 | 6.25 | 6.33 | 5.71 |

### 5.3. Statistical test setting

The tests in section 5.1 were performed again but evaluated in a statistical way. It is actually difficult to perform a statistical test. This is that our machine re-target a performance trend rather than directly copying each dynamic level, by smoothing every vector before training. Therefore the original performance will not be reproduced even though the training data and predicted data are the same, and hence it is difficult to do a statistical test to fully evaluate the performance of the machine. However, it is still possible to evaluate the machine's performance in a statistical way by comparing the cluster code between the output files. Moreover, we can calculate the run time to evaluate the efficiency.

### 5.4. Statistical test result

The result of the statistical test is shown in Table 2.

Table 2. The s*tatistical test result.*

| | Test 1 | Test 2 | Test 3 |
|---|---|---|---|
| **Accuracy** | 93.2% | 91.7% | 68.2% |
| **Run Time** | 42.43s | 67.21s | 58.05s |

First of all, the run time of the machine is as expected and very efficient. Test 2 has the longest run time because it has a larger training set. We will keep working to shorten the run time.

Both tests 1 and 2 show excellent results, proving that the SVM feature vector design is appropriate, where test 2 shows that excess training data will not affect the accuracy, and nearly no input vector will run into an unseen situation.

The result of test 3 is also as expected and does not mean the experiment was unsuccessful. There are a lot of limitations in performing statistical tests for test 3. The most important reason is that many similar dynamic trends will not merge into the same cluster. As described in section 6.26, we have tight criteria for cluster merging. A loose threshold will result in very few clusters, where all will being plain, uninteresting and unexpressive. When we carefully looked into some incorrectly classified vectors, we found that they were actually very similar but belonged to different clusters. For example, some of them only had different ending dynamics, while some had a sudden raise or drop of dynamics in the middle. However, the rest of the clusters were almost the same. Actually the performance of this machine is difficult to describe in a mathematical comparison. However, it does reflect the machine's ability to certain content.

## 6. FUTURE WORK

Here is a list of the proposed future work.

### 6.1. Feature vector with more parameters

Research on articulation parameters is the first priority to be settled among all the future work. It will be used to build up a 2-D performance trend with the z-score dynamic trend. One of the main problems needed to be solved is how to measure note ends. More research on human perception will be done.

### 6.2. Global trend ratio

The global trend ratio is currently just a single number. It should be possible to describe it as a formula, for example, in the form of a regression or polyphonic equation.

### 6.3. Work on Polyphonic music

The current machine is actually optimized for a polyphonic approach already. The only remaining problem needed to be solved is melody extraction. This topic is highly problematic and needs another full paper to discuss it. However, we will try to find a robust and efficient approach which extracts pitch only. We will discard the MFCC and tone features. This should be adequate and feasible for this machine.

## 7. CONCLUSION

Building computers that can learn musical performance style has been a hot topic in the field of artificial intelligence. In the past few years, research in AI and music has been creating systems that mimic human perception in order to recognize musical structures like a trained musician. Previous experiments show that it is possible to classify music into genre by learning; hence there exists some common style in music of the same genre. The next question is whether it is possible to extract music performance style parameters and reproduce expressive musical performance through a black box.

For completely automatic conversion of expressive music, while there has been some success in specialized problems such as beat tracking, most truly complex musical capabilities are still well outside of the range of computers, for example, identifying form and motif structure. From a practical point of view, the current technology is not advanced enough for the computer to understand music as a professional musician does, but it is intelligent enough to help and support musical applications. The automatic production of expressive music at present still requires human intervention in some form.

In the future, we will continue to work on increasing the accuracy and enhancing the run time of the program. For accuracy, more research on human perception and more observations of articulation parameters from real recordings will be done. For the run time, the workflow will be further optimized and we will find a more simplified approach which does not affect the accuracy. Our experiment already shows that it is possible to induce some of a performer's style. This is a stepping-stone for the next stage of our research.

## 8. ACKNOWLEDGEMENT

This work was supported by the RGC grant 613508 and the RTG Travel Grant.

## 9. REFERENCES

[1] P. Zanon, G. Widmer. Recognition of Famous Pianists Using Machine Learning Algorithms. XIV Colloquium on Musical Informatics (XIV CIM), Firenze, Italy, 2003.

[2] G. Widmer, S. Dixon, W. Goebl, E. Pampalk, A. Tobudic. In search of the Horowitz Factor. AI Magazine, 2003

[3] S. Dixon, W. Goebl, G. Widmer. The Performance Worm: Real Time Visualisation of Expression based on Langner's Tempo-Loudness Animation. International Computer Music Conference, Göteborg, Sweden, pp 361-364, 2002.

[4] G. Widmer. Using AI and Machine Learning to Study Expressive Music Performance: Project Survey and First Report. AI Communications 14(3), 149-162, 2001.

[5] S. Dixon. Automatic Extraction of Tempo and Beat from Expressive Performances. Journal of New Music Research, 30 (1), pp 39-58, 2001.

[6] C.-W. Hsu, C.-C. Chang, C.-J. Lin. LIBSVM: a library for support vector machines. National Taiwan University. 2004.

[7] G. Peeters. Music Pitch Representation by Periodicity Measures Based on Combined Temporal and Spectral Representations. IEEE International Conference on Acoustics, Speech and Signal Processing, 2006.

[8] S. Lui, A. Horner, L. Ayers. An Intelligent SP-MIDI Polyphonic Reduction Algorithm. IEEE Transactions on Multimedia, Volume 13, Issue 2, pp.52-59, 2006.

[9] C.-W. Hsu, C.-C. Chang, C.-J. Lin. A practical guide to support vector classification. National Taiwan University, 2004.

## 10. APPENDIX I: AN OVERVIEW OF OUR IMPLEMENTATION