Musical Note Identification
Violin Score Reproduction from Audio Recordings

Ross Kidson ROSS.KIDSON@GMAIL.COM
FangYi Zhi FANGYI.ZHI@TUM.DE
Department of Computer Science, Chair of Robotics and Embedded Systems, Technical University Munich Boltzmannstr. 3, 85748 Garching bei Mnchen

Abstract

Pitch identification is a difficult problem that has many potential applications including music transcription, music recording and speech processing. There is no one single optimal approach to this problem, rather various approaches that have individual strengths and weaknesses. The aim of this project is to develop a machine learning algorithm to detect musical pitch. This algorithm has been developed specifically concentrating on the violin, however this approach could also be applied for other instruments. The overall goal of the project is to develop an algorithm that is able to produce a kind of script given the recording of a song.

The spectrogram method was used on sound recordings to generate features. Logistic regression and Support Vector Machine classifiers were identified as suitable learning algorithms for the problem. A discussion of the training data, test data, choice of features, implementation of algorithms and obtained results will be presented. It was found that logistic regression was showed very good performance, whereas support vector machine did not show such good performance.

1. Important terms

1.1. Theory Introduction

When an instrument sounds notes, the sound being heard is not only one frequency, rather a combination of **overtones**, or **harmonics**. The degree that these different harmonics come into the sound play a critical role in defining the sound or timbre of the instrument.

The **Fundamental Frequency** is the lowest frequency of a note, this is also the frequency that corresponds to the pitch that is identified by a human listener. All harmonics are a multiple of the fundamental frequency.

There are a total of 12 different notes. These notes repeat themselves over different octaves. Going up one **octave** means doubling the frequency. In this algorithm the octave is considered as well as which note, meaning there are much more than 12 classes.

1.2. Class definition

The playing range of violin is theoretically from G3[1] to C8[2]. But for the normal usage, the highest note can be set to G6. Since every octave has 12 semitones, 38 classes were defined from G3 to G6 according to the chromatic scale. The following picture shows class labels in respect of its musical pitch.

The goal is to classify every note from a violin audio record into one of these thirty eight classes and generate the music score in the end.

2. Data Collection

2.1. Recording setup

The recording set up was very simple, a standard pc headset microphone was used to directly record the violin. All data used was from these recordings. The headset was worn by the player as to position the microphone as close as possible to the violin to reduce noise. Various sampling rates were tried out however

Appearing in *Proceedings of the 27th International Conference on Machine Learning*, Haifa, Israel, 2010.

[1] The G below middle C
[2] The highest note from the modern piano.

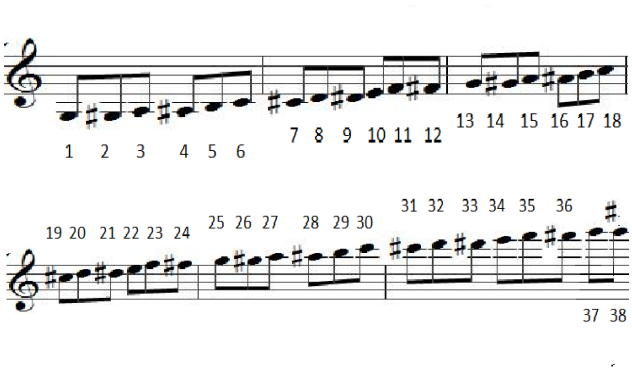

Figure 1. Class labels with their corresponding musical notes

44.1khz seemed to provide a good compromise between quality and data/memory requirements.

2.2. Training data

Producing training data that allowed for good results in the machine learning algorithms provided a significant challenge in this work, and many recordings were made before a good set was generated.

From literature it has been noted that using normal music is a potential source for data. The approach in this case is to split the musik up into small time segments and classify each segment. The advantage of this is that a single piece of music can potentially provide many training examples based on the segment size. In addition this is an accurate representation of the data that later will need to be classified, which variations in playing style and sounds from the instrument that may not contribute to the pitch information. On the other hand this kind of data can be detremental to training due to the introduction of large amounts of noise. In addition it is also neccessary to label all the time segments, which requires either significant time or some automated approach, which was not favoured in this case.

Another possibility of for training data is to record the notes individually in a controlled environment. This was the prefered option due to the simplicity of labelling, and the ability to produce clear, simple training data, which was shown to provide good results. Using the simplified "clean" training data to obtain good results may have come at the risk of lacked flexibility, however was not further investigated.

2.3. Testing data

As described above, the goal was to classify songs and produce scores. Therefore, for the testing data a number of differing songs were recorded. This data needed to in turn be labelled, which was performed manually by hand. These labels were then interpreted by some matlab code and compared with the algorithm output in order to give us an approximate accuracy value. As the labelling was done manually, there is an error to consider in note bounderies. In the end 2 labelled songs were used for testing, the Secret Garden, a slower song and First Concerto from Bach, a faster, more technical song.

There was a memory issue with importing longer sound files into octave, and therefore all test songs were about 30 seconds long. The Bach peice has been split into 2 different examples.

3. Features

3.1. FFT

Fast fourier transform is an algorithm to efficiently calculate the discrete fourier transform of a signal. The discrete fourier transform decomposes a signal down into different frequency components, showing the relative magnitude of each frequency on the signal. This is a perfect tool for deciferring the various harmonics of a signal as mentioned above. See figure 2. A number of possible features from the FFT are as follows:

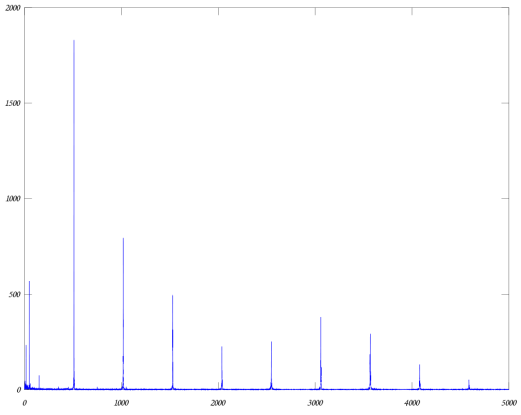

Figure 2. Discrete Fourier Transform

- The entire FFT (a feature for each frequency)
- The frequencies of the peaks

- The relative magnitudes of the peaks
- The fundamental Frequency
- Distance between peaks

All of these methods were tested for suitability. It was found that the entire FFT amounted to be too many features for an efficient operation. To simplify this it was thought of to define features as peaks, and assign either frequency values or relative magnitudes to each feature. However using the frequencies of harmonics seemed rather redundant as they were always a multiple of the fundamental frequency. Another option was just to take the fundamental frequency as a single feature. Although this enourmously simplifies the problem, it is not always so easy to identify the fundamental frequency, especially when there is other noise in the signal.

Having tested a number of the potential features it was found that the classes were separable, and it was possible to classify notes based on the FFT taken from recordings of single notes. However the goal is to be able to classify a the notes in a song, also specifying the timing of the notes. The intended approach is to break the recording up into small time segments and classify each segment. Whilst it is possible to do this with the FFT, a much more elegant solution exists to perform a DFT over time: the spectrogram.

3.2. Spectrogram

Spectrogram, or STFT, essentially provides an DFT over time. The typical spectrogram graph has x axis as time, y axis as frequency and a third vale, shown as intensity, to describe the relative frequency amplitude at a given time. This is calculated by splitting the signal into time segments with overlap, performing FFT on each segment, and then combining all FFTs in a single graph. This method of feature extraction is ideal for the approach of classifying time segments to produce a score, however there is one drawback of the spectrogram, namely find the compromise between time or frequency resolution.

By having a set window size for calculating FFTs, the information being provided for FFT calculation is limited. By choosing a large window a FFT with very good frequency resolution is obtained (und thus good seperability of notes). However this will result in large overlap between notes, and degrade the FFT on note boundaries, as the two notes will be combined together. On the other hand, by choosing a small window, good seperation between notes can be obtained, as the overlap between notes is reduced, but

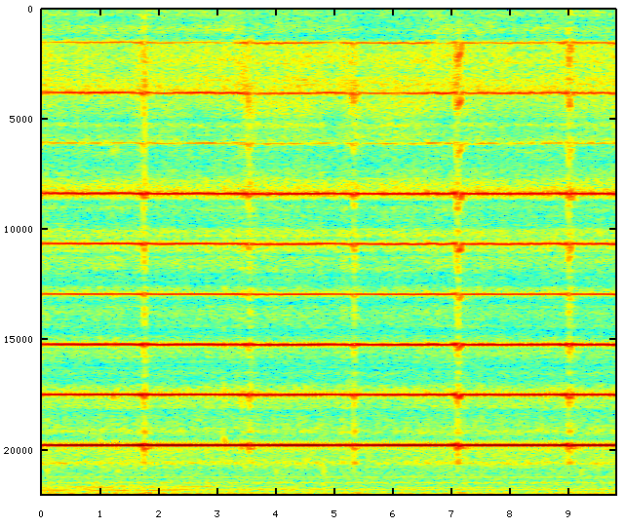

Figure 3. Spectrogram of a violin playing a single note for 10 seconds

the frequency resolution is compromised, and possibly the ability to separate notes. The compromise in frequency resolution can be seen in fig.4 Therefore this window size is a critical parameter in tuning the learning algorithms.

With this data, it is also possible to identify the fundamental frequency for every time unit, and use this as a feature. However, as per the stadard DFT, this is not so reliable and with a reduced frequency resolution the seperability of notes is not so good, especially for lower notes. In addition, as mentioned above, the relative magnitude of frequencies is also somewhat characteristic of a note and can be used to assist classification.

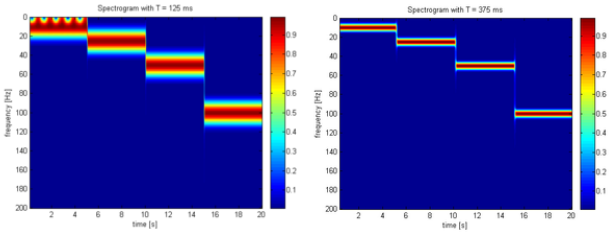

Figure 4. Spectrogram frequency resolution compromise with differeing window sizes

Therefore to incorporate this information into the features, the entire "spectral slice", being a vertical line across the spectrogram for a given time, was used as a single training example. There was then a feature for every frequency intensity. The strong advantage

of spectrogram as feature identification in this application was that it was easy to change the frequency resolution through the parameters of the spectrogram, and thus the number of featues, allowing for tuning of machine learning algorithms. For these reasons this approach was selected for obtaining features for both machine learning algorithms.

An attempt was made to visualize these features and see if the notes are separable. Here is the PCA visualization from the 742-Demensional feature space generated from spectrogram to a 3-Demensional space. Some classes are obviously separable, however it is hard to visualize, particularly with so many different classes.

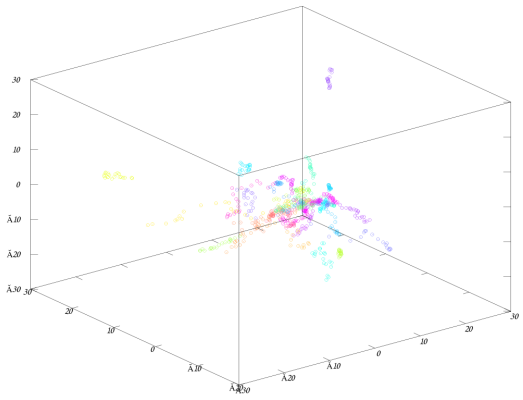

Figure 5. PCA visualization of 38 classes with different colors

4. Logistic Regression

4.1. Implementation

A one vs the rest Logistic Regression Classifier was utilized in this approach. This was implemented in octave. This was selected because of its simplicity and wide use. As mentioned above the features were taken from the spectrogram, the training examples being individual time slices. The window frame was experimented with, optimal values were in the 10 - 100ms range, providing between approximately 600 and 4000 features.

4.2. Results

The Logistic Regression algorithm provided overall very impressive results. In figure 6 and 7 the output of the classification for the first part of the selected Bach piece can be seen. Red is the correct pitch, and Blue is the output from the algorithm. Figure 6 shows the original output of the algorithm, where 7 shows a filtered version of this. The filter removes misclassify peaks by checking before and after note values over a small change in time. Although this looks like a vast improvement, it only improves the overall accuracy by about 2% which means these peaks are a minority of the data. This filter is only to help visualize the data, results are quoted without the use of the filter.

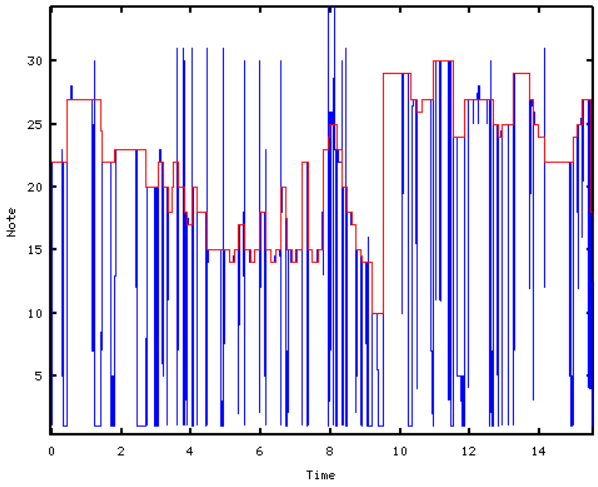

Figure 6. Output of algorithm without filtering

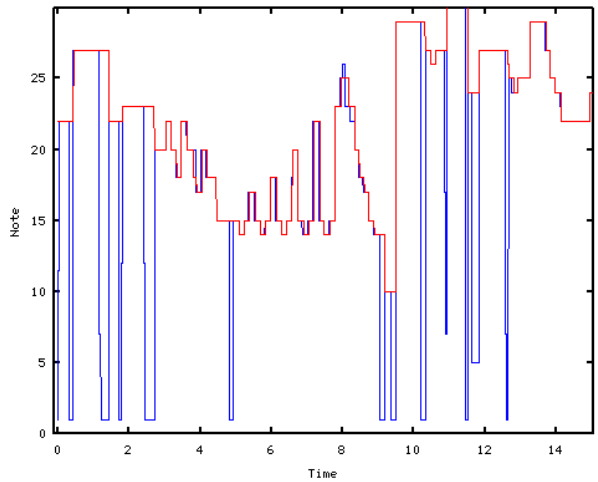

Figure 7. Output of algorithm with filtering

As discussed above, the window size was a critical parameter in tuning the performance of the algorithm. In table 1 results for the different songs and different

Table 1. Performance of Logistic Regression with varying window size

Data set	10ms	30ms	50ms
Bach 1	72.9%	77.2%	66.3%
Bach 2	76.8%	76.7%	59.3%
Secret Garden	92.0%	93.6%	80.9%

window sizes can be compared. On the slower song it can be seen that very good performance is obtained. Furthermore, it is clear that a 30ms window size is optimal. Regularization was also considered as a parameter to optimize the performance, however introducing this did not show improvement and was therefore discarded.

5. Support Vector Machine

The open source machine learning library **libsvm** was used to implement a one vs. one support vector machine on the training data. The algorithm was developed using the following procedure:

- Data Scaling
- Using RBF Kernel $K(x, y) = e^{(-\gamma||x-y||^2)}$
- Using Cross Validation to find the best parameters C and γ
- Using the best γ to train the whole training set
- Test with the test Data

It was found that the performance didn't change a much by increasing the number of training examples. Therefore the number of training examples was kept the same in the following experiment.

After determining the learning curve, the number of features was chosen to be 184, which is corresponding to a window size of 10ms.

5.1. Parameter Selection

5-fold Cross Validation was used to choose the best parameters C and γ. 5 logarithmically spaced values of C were used and 5 logarithmically spaced values of λ were used. A better analyse could be given by choosing a larger number of parameter values, but due to the expensive computation time caused by the high dimensional feature space and large number of training examples, it was decided to keep the current parameter values.

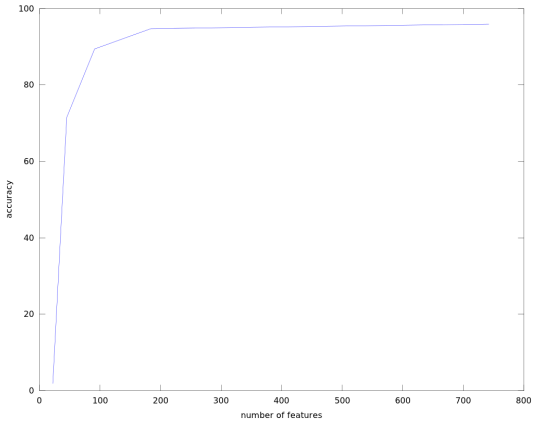

Figure 8. Learning Curve of SVM

Table 2. Performance of SVM

Data set	10ms
Bach 1	43.7%
Bach 2	41.9%
Secret Garden	46.7%

After the parameter selection precedure, the best values of C and γ were obtained with an average accuracy of 95.91%. The test data was tested with the parameters $C = 1$ and $\gamma = 2^{-5}$,

5.2. Test

The results of the test data was showed in the table.

These result are significantly worse than the results of Logistic Regression. The main reason could be that the number of classes is too large for an SVM to perform well. Due to the mechanics of SVM, increasing the number of training examples doesn't improve the performance of an optimized SVM, the number of support vectors stay approximately the same.

6. Comparison of Algorithms and Conclusions

Logistic regression showed very good performance on the data, providing relatively accurate results even on faster music, and given the restriction of the time/frequency resolution compromise. It is rather strange that the SVM algorithm did not perform at least as well at logistic regression. It had performed quite well in the cross validation of the traning data,

but was never able to produce decent results on real test data. A theory for this is that the Support Vector Machine does not work so well with such a large number of classes.

Another aspect to consider in the comparison between logistic regression and support vector machine is that logistic regression did not perform as well with less training examples, and thus it was important to provide enough training data to achieve the accuracy obtained. However Support Vector Machine did perform well even with less training examples. This may well be a the advantage over the Logistic Regression, however overall performance was never comparable between the algorithms, so it is difficult to make this conclusion.

Given more time, it would have been good to further troubleshoot the SVM algorithm, as it should show at least comparable performance. On the other hand this also shows the simplicity and power of logistic regression.

References

1. G. Poliner, D. Ellis, A. Ehmann, E. Gmez, S. Streich, B. Ong (2007), Melody Transcription from Music Audio: Approaches and Evaluation, IEEE Tr. Audio, Speech, Lang. Proc., vol. 14 no. 4, pp. 12471256, May 2007,.