Generating Sheet Music From Audio Files

Jan Dlabal and Richard Wedeen

Stanford University

jdlabal@stanford.edu, rwedeen@stanford.edu

December 13, 2013

Abstract

This paper describes a method for using supervised learning to generate sheet music from audio files of single instrument melodies. Support Vector Machines were used to detect the presence of up to 88 notes with one-versus-all classification. We also outline preliminary steps toward transcribing a whole melody by using a sliding window to detect note changes in an audio sample.

I. Introduction

Many amateur musicians would like to be able to play popular songs, such as those they have in their iTunes library. However, the only way to obtain sheet music for a song is to buy professionally made copies, which are often either expensive or unavailable. This creates a barrier for people to truly enjoy their music and confines them to listening, instead of playing, the songs they love.

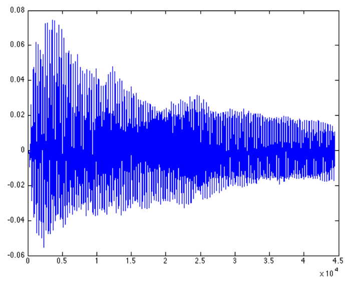
To solve this problem, we explored using supervised learning to generate sheet music from audio files so that musicians can obtain sheet music quickly and cheaply. We focused on generating sheet music for single instrument melodies by breaking the problem into three stages: single note classification, transition probability markov models, and note-change detection.

II. Single Note Classification

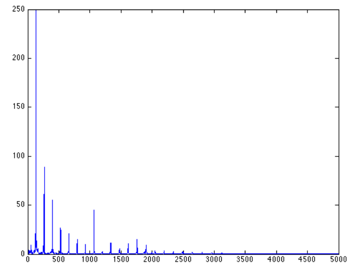
In single note classification, we trained multiclass SVMs with Gaussian Kernels in one-versus-all classification for checking whether a given note is present in the input, which included single notes and major/minor chords.

I. Data

We generated individual notes and major/minor chords from MIDI files (computer representation of sheet music) using an assortment of 20 soundfonts.



(a) *Time Domain*



(b) *Frequency Domain*

Figure 1: *FFT of the raw input data*

A soundfont is a file that a synthesizer uses to generate actual raw samples from the MIDI files giving the notes that should be played. This gave us around 1000 WAV files each for single notes, major chords, and minor chords.

Figure 1a shows an example of a generated WAV file from the combination of a sound font and a MIDI file representing a single note. We then applied a Fast Fourier Transform to each sound file (see figure 1b). This was then "binned" to reduce overfitting; bins grouped multiple adjacent frequencies as shown in figure 2 below. These binned frequency vectors were then used as learning features.

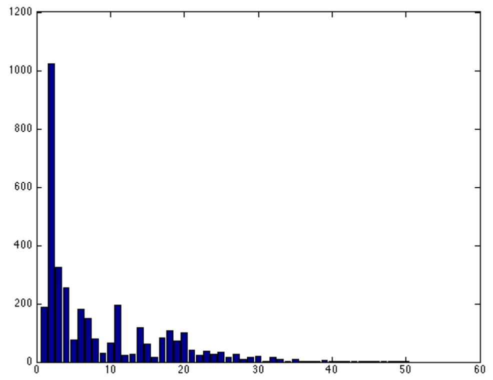


Figure 2: *binned fourier frequencies*

Finally, we randomly separated the input data into a training group and a testing group that contained 70% and 30% of the data, respectively.

II. Training

Each sound file was labeled as a binary valued vector with ones indicating which notes were in the file. For example, a C major chord starting on middle C (i.e., the set of notes $[C, E, G]$) would be a vector containing 1s at indexes 60, 64, and 67 (see Figure 2). Similarly, C minor would contain 1s at indexes 60, 63, and 67. Finally, a WAV file containing only C would be represented by a 1 only at index 60.

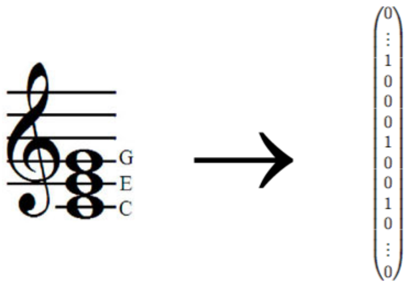


Figure 3: *Binary vector representation of a C major chord*

We then trained SVMs for every note with the training data. For example, the SVM for middle C was trained to predict whether or not a sound file contained middle C. This means that it should predict a 1 even if it was fed an A minor chord, because that chord consists of the notes $[A, C, E]$ which contain C.

III. Classifying a single note

The negative class of our SVM corresponded to an input that contained the note, and the positive class corresponded to an input that did not contain the note. To classify a new note, the SVM with the most negative score was selected. This corresponds to picking the lowest possible $\gamma(x_{new})$ as the most probable note, where x_{new} is a new example.

III. Note-Change Detection

I. Data

We generated single-instrument WAV files of melodies from midi-file/soundfont combinations.

II. Method

Note-change detection aimed to find the points that notes changed in a melody. The goal was to produce equal-note intervals in the sound file that could be fed into the single note SVM classifier.

To detect note changes, a sliding window was used to sweep across the melody audio file. At each instant, single-note detection was applied to the window and the functional margins from the SVM were recorded. The minimum of these margins corresponds to how "confident" the SVM was in predicting what the note was. Windows that contain more than one note value will cause the SVM to be less "confident", and output a minimum margin value that is greater than the minimum margin of a window that contains only one note value.

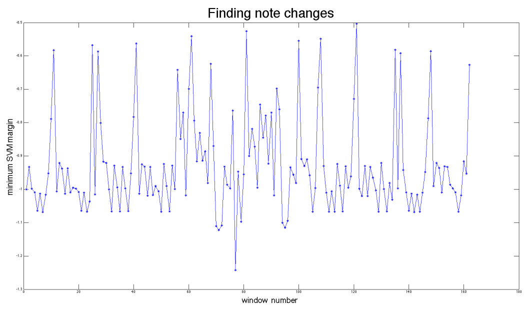


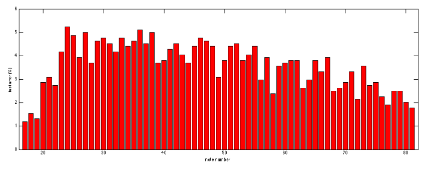
Figure 4: *Minimum Margin Values*

If the minimum margin value at a window value was less than the previous, then it was deemed to be more "confident" than the previous and our algorithm marked the window value as a new note.

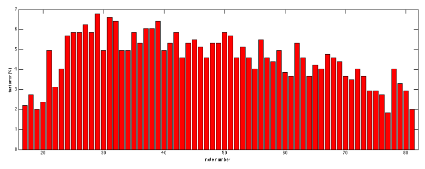
IV. Results

I. Single Note Classification

The cross validation test errors on the test set (single notes and chords) are displayed below (figure 4). The test error for note j corresponds to the percentage of sound files that the SVM correctly predicted whether or not it contained note j. Figure 4(a) corresponds to the test error when using SVMs that were trained on all combinations of major/minor chords and single notes, whereas figure 4(b) corresponds to training only on major/minor chords with the testing is still done on per-note basis.



(a) *single notes and major/minor chords*



(b) *just chords*

Figure 5: *70/30 Cross Validation Errors*

Single-note classification on a melody line with pre-specified note-change values yielded 97% accuracy.

II. Note-Change Detection

Running note-detection, identifying equal-note windows, and performing single-note classification on a melody genererated from a midi file yielded a sequence of notes that shared a longest common subsequence around 86% as long as the original (correct) sequence of notes. Different sliding window lengths yielded different accuracies, but they were all around this value (see figure 5).

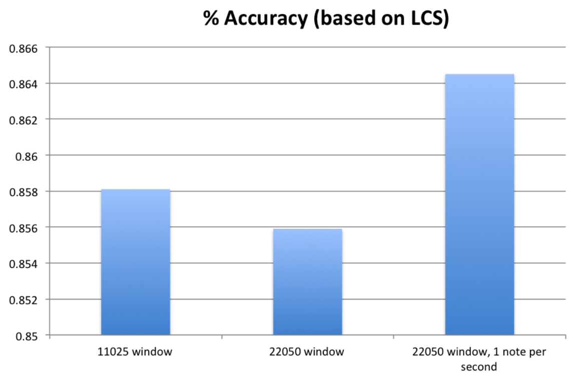


Figure 6: *note-change detection accuracy on melody*

The final results are summarized in the table below. Isolated corresponds to the case where the note changes were pre-specified. Melody corresponds to the unspecified case.

Table 1: *Accuracy of Melody Detection*

Method	Accuracy
Isolated	97%
Melody	86%

V. Discussion

The single note classification performed surprisingly well on the given dataset of a wide variety of electronic instruments. This could be attributed to the fact that there was no noise present to corrupt the data, something that would be common in a real-world example. We suspect that the classifier peformed better on the lower range notes (see figure 4) because some instruments were incapable of producing these low registers and consequently these SVMs could tailor more specifically to this smaller subset of instruments.

Note-change detection worked reasonably well given the simplicity of the method. It might be improved by noticing that in its current state the method is susceptible to false positives since random fluctuations may cause a margin minimum to be less than the previous, signalling a change. A similar approach worth exploring would be to check if the minimum has dipped below a certain threshold to signal a note change.

VI. Conclusions and Future Work

Overall, this approach provides a good start to generating sheet music from audio input. However, it only considered the case of a single instrument and avoided the much hairier problem of separating out multiple instruments. For audio files with sufficiently different left-right stereo output, the separation task can be simplified by performing independent components analysis. Unfortunately this will only simplify a small subset of music files.

Future work should focus on this problem as well as testing how well this method works when the signal-to-noise ratio is low.