

Automatic Transcription of Solo Piano Music

Alex Landau
CS 229, Autumn '09

Music transcription is the process of recovering the pitches and timings of notes in a piece of music from a recording of the music. This can be accomplished by a trained human musician, but automated music transcription, performed entirely by computer, has been a difficult task. For this project, I am exploring the problem and the potential applications of machine learning therein. I have examined the problems reducing the effectiveness of other groups' work on the subject and developed a structure for a new algorithm that can eliminate these problems.

One straightforward application of machine learning techniques to this problem was done by Graham Poliner and Daniel Ellis of Columbia University.¹ Their approach used one-versus-all SVMs to train one classifier for each note; the features from the SVM were short-time Fourier transforms from a spectrogram. This gave them estimates of the p_i , which they used to estimate the probability that the prediction is correct. This was then followed by a Hidden Markov Model application to smooth the results for each note over time. Both classifiers were trained on a large array of sample music.

This paper also provides a methodology for determining the error produced by a transcription algorithm. It separates the music into short time frames on which it performs its predictions, and it matches this against an already-calculated set of data. The errors are further classified into substitutions, misses, and false alarms. They also suggest another error measure, involving note onset detection; this avoids problems associated with determining the endpoints of notes that fade out over time, as often happens in piano music.

Selecting appropriate features is crucial for note recognition. Raw audio data comes in the form of numbers representing the amplitude of the sound wave at various points in time. (This is the format of .WAV files; .MP3 files have the same data, but in compressed form.) One very useful technique for analyzing this data is the short-time Fourier transform, or STFT. This transforms the sinusoidal data in a short period of time from the amplitude space to a frequency space. Furthermore, the STFT can be applied repeatedly to overlapping segments to form a spectrogram, which shows the strengths of each frequency over time. (MATLAB's Signal Processing Toolbox provides this as a feature.) The spectrogram can be inverted to retrieve the original audio data; this is a guarantee that it contains enough data to perform transcription.

The output of the STFT is a series of complex numbers. The absolute values of these coordinates reflect the strength of the sinusoidal signal of the given frequency; the complex number's angle in polar form represents the specific phase of the signal. One of my experiments regarding the spectrogram data involved removing the phase from this data to see if information needed to transcribe the notes would be lost. This involved generating a spectrogram of some sample audio data, modifying the spectrogram, and inverting it to retrieve the equivalent audio data. (This was accomplished with the use of a spectrogram inverter written by Daniel Ellis). In the resulting audio file, the original notes were clearly audible and distinguishable, so there is sufficient information to perform audio transcription if phase information is ignored. The remaining amplitude information (represented entirely by positive real numbers) is easier to work with and visualize.

One feature that is peculiar to the piano can help in transcription: the noise produced by the hammer when a new note is struck. This sound is clearly distinguishable from that produced by a vibrating string: it fades as quickly as it rises, and (more importantly) it is spread across a broad

¹ Poliner, Graham E. and Ellis, Daniel P.W. "A Discriminative Model for Polyphonic Piano Transcription." *EURASIP Journal on Advances in Signal Processing*.

spectrum of frequencies. As a sound with these properties appears at the beginning of each new note, it can be used to determine when new notes are struck. A simple version of this function is as follows:

```
function hammerTimes = getHammerTimes(S,threshold)

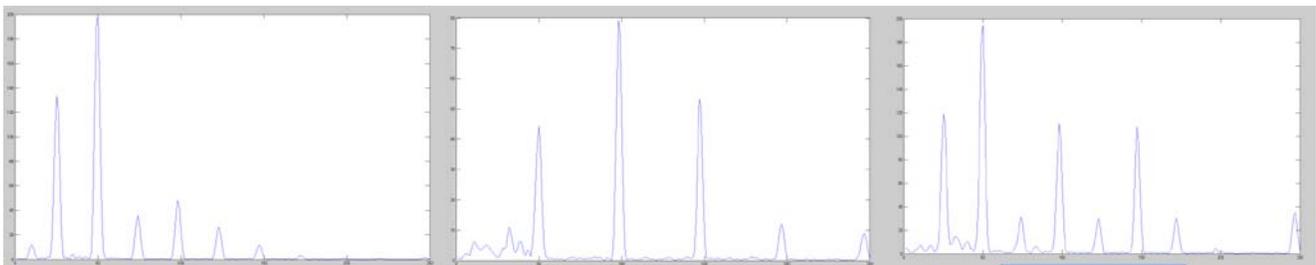
t = size(S,2);
hammerTimes = find(sum(S(:,2:t)-S(:,1:(t-1))),1) > threshold) + 1;
%This gives the leading edge of each hammer strike; the last one should be
%in roughly the middle of the hammer strike.
hammerTimes = keepLastOfEachGroup(hammerTimes); %Keeps the last of each series
%of consecutive integers
```

This version was tested on three different sample audio clips, two recorded by myself and one professionally recorded. In all three, its threshold parameter could be set to accurately list every time that a beat was beginning and nothing else; unfortunately, no single threshold was optimal for all three. It may be possible to determine the correct threshold using other features of the music. Even if the parameter cannot be automatically set for the piece, it is a useful input for determining the points in time likely to be the beginnings of notes. (The difference in total amplitude taken by itself is also a useful input, but this method has the advantage of narrowing the initial time frame to a single value.)

It would be logical that other versions of the function taking advantage of how hammer noise is spread across the frequency spectrum would be even more effective; however, my attempts at incorporating this effect have failed. Part of this may be that this particular implementation also incorporates the increased noise from strings, also marking the beginning of a new note.

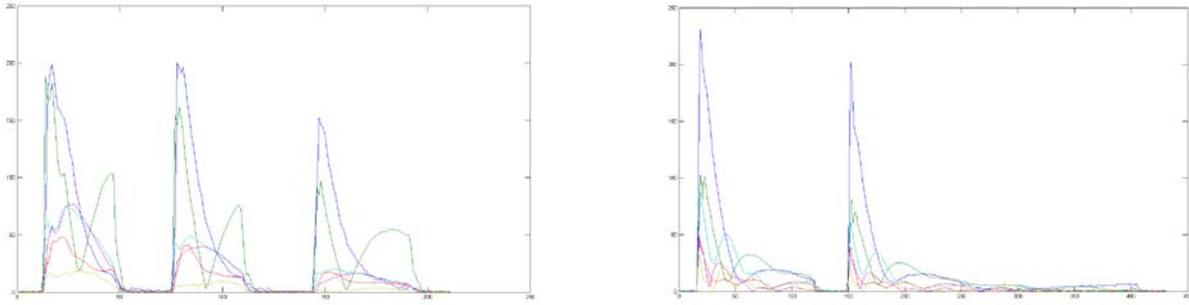
The problem that arises with the SVM method described above is that it frequently makes mislabelings in which the labeled and correct notes are off by an octave or similar interval. To see why these intervals are difficult to distinguish, we need to consider the forms the spectrogram data takes.

Each note played during a timeframe contributes its own fundamental frequency, but it also contributes additional frequencies that are integer multiples of the fundamental. These can be seen as Gaussian spikes in the frequency chart. This makes it easy to distinguish individual notes based on which spikes are present. However, when there are multiple notes and one's fundamental frequency is a multiple of the others, the spikes will overlap, hiding the second.



The Gaussian spikes generated by one note; a second note an octave higher; and the interval containing both.

To distinguish these two cases, we need to know more about the heights of the spikes relative to one another and how they differ in those cases. Looking at several examples, we see that the changes of amplitude of each spike in a given note are predictable, even when keys are played with different weights.



Middle C evolving in time on two different pianos. Each colored line represents a different Gaussian spike.

One way of incorporating this data would be to add time as a component of the SVM's feature vector. Specifically, we could add to the feature vector a parameter consisting of the time since the beginning of the note (times some constant factor to make it more relevant in computing the distance, as needed). This would be easy to incorporate when training the SVMs, as this can be computed from the data set. (The negative samples would need to have this parameter filled in according to some distribution matching that of the positive samples.) The problem comes in applying the SVMs to real data: we no longer know what offset to use as a feature. The solution is to make use of hammer noise. If we have some set of times, a subset of which are the correct hammer strike times, then we can apply our SVM test once for each recent hammer time. A note can be considered likely to be played during a given time frame if it passes an SVM for any one of its likely time parameters.

The problem with this improvement is that its usefulness is severely restricted by its need for training data. As mentioned before, note variations over time are instrument-specific, so the training data must come from the instrument to be tested for full usefulness. Matched recordings and note data are only practical to acquire with synthesized instruments, and even then only with access to the original synthesizer. Therefore, it would be preferable to have a method that learned from the unlabeled recording itself.

We can envision the problem in terms of likelihood maximization under an EM approach. In this case, the hidden labeling is the set of notes that are played at various times with various amplitudes. A labeling consists of the number of note assignments along with each assigned note's pitch, amplitude relative to the model's, and timing.

To reduce the complexity of the problem, the E-step only tries to propose one labeling, instead of weighing multiple possible labelings. A solution to this problem is very difficult, but feasible. In theory, it could be solved by treating it as a Hidden Markov Model problem. Each timeframe has a labeling. Most of the transition probabilities will be zero; if one time frame has a given pitch being played, the next time frame can only contain that pitch being played in the subsequent time step with the same relative amplitude or not being played at all. When a pitch is not being played, it can either continue to not be played or be played with some amplitude (discretized for the purposes of the HMM).

A suitably optimized Viterbi-like algorithm, taking advantage of the high likelihoods of maintaining the status quo and of playing small numbers of notes as opposed to larger ones, might be able to solve this in a reasonable amount of time. I have instead pursued a different algorithm that takes advantage of a certain property of the notes: the lowest frequency can only be contributed by its own note. (Unfortunately, given my current treatment of the music, this property breaks down in the lowest octave or so, where the "lowest frequency" of each note is too close to distinguish. This could be addressed by using a broader range of spectrogram data: a larger STFT window would allow greater resolution at these low frequencies, at the expense of precise time measurements.)

Notation:

$x^{(t)}$: Observed frequency vector of a timeframe t

$\theta_n^{(r)}$: Proposed distribution of the frequency vector of note n at time r with respect to the beginning of the note

ε : Distribution of background noise (half-normal distribution)

s_i : Starting time of assignment i (measured in time frames)

$a_i^{(t)}$: Relative amplitude of assignment i if it is played during time t ; otherwise, 0.

n_i : Note (i.e., pitch) of assignment i

$\varphi^{(t)}$: Predicted distribution of the frequencies at time t , given the assignments

Then an additive model gives $\varphi_j^{(t)} = \sum_i a_i^{(t)} \theta_{n_i j}^{(t-s_i+1)} + \varepsilon_j$. However, this model gives incorrect

results because it does not take into account the effects of destructive interference. Each note contributes to a given frequency some complex value representing both its amplitude and its phase; the contributions are added in this complex space, so they are as likely to cancel each other out as they are to add perfectly.

We can create a reasonable model for this interference if we make the assumption that one note has a significantly higher mean value than the others. Suppose that we have random Gaussian variables A and B , and we want to find the distribution of $C = |A \cos \varphi + B \sin \theta|$, with φ and θ uniformly and independently distributed. (This reflects how the waveforms are added.) Under the assumption that $A > B$, we find that this is equivalent to $C = \sqrt{(A + B \cos \theta)^2 + B^2 \sin^2 \theta} = \sqrt{A^2 + 2AB \cos \theta + B^2}$, which is closely approximated by $A + B \cos \theta$. This in turn has expected value μ_A and variance $\text{var } A + \text{var}(B \sin \theta) = \sigma_A^2 + E[B^2]E[\sin^2 \theta] - E[B]^2 E[\sin \theta]^2 = \sigma_A^2 + \sigma_B^2(2 + 4\mu_B)(0.5) = \sigma_A^2 + \sigma_B^2(1 + 2\mu_B)$. Adding background noise gives variance $\sigma_A^2 + \sigma_B^2(1 + 2\mu_B) + \sigma_\varepsilon^2$. Then, our estimated distribution for

$\varphi_j^{(t)}$ becomes $N\left(\mu_{m_j}^{(t-s_m+1)}, a_m^{(t)} \sigma_{m_j}^{(t-s_m+1)} + \sigma_\varepsilon^2 + \sum_{i \neq m} \sigma_{ij}^{(t-s_i+1)} (1 + 2\mu_{ij}^{(t-s_i+1)})\right)$, where $m = \arg \max_i a_i^{(t)} \mu_{ij}^{(t-s_i+1)}$.

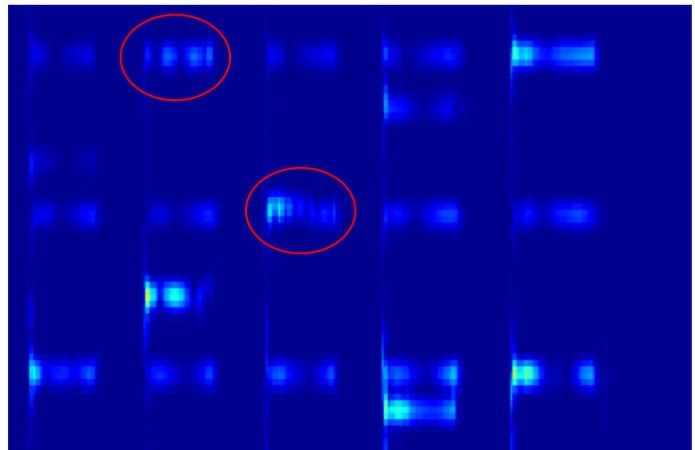
The likelihood to maximize is $L(\theta) = p(\theta) \prod_t \prod_f p_{\varphi_j^{(t)}}(x_f^{(t)})$.

Is our assumption about the relative values of A and B practical? In fact, it describes most cases, particularly because the parameters vary both with frequency and with time. Rarely do the curves of the multiple notes come close to matching; and when they do, the variance is high, so they contribute little doubt to the model if the m chosen is incorrect.

In practice, when two notes contribute the same frequency at the same time, this leads to the phenomenon of beating. This happens when the frequencies from the two notes are not quite the same. As a result, the amplitude of the frequency in general (its associated spike) oscillates over time. This produces a good indicator to the algorithm that the frequency is best explained by a high-variance combination of notes, rather than a single note.

Examples of beating as seen on a spectrogram.

The M-step of the algorithm assumes that our assignment of notes is correct and subsequently tries to maximize $\ell(\theta)$ by changing the values



of . If we allow arbitrary iterated maximization this way, then we could potentially end up with note forms that are dissimilar from the intended notes, or from any actual notes; to prevent this, we apply a Bayesian prior to each θ_n . (An alternative would be to allow arbitrary evolution of the θ s, then try to distinguish the notes coming from the resulting evolutions; but this could misclassify octaves as notes, for example.)

The prior probabilities require some knowledge about how the notes are distributed. The fundamental frequencies are easy to calculate, due to even tempering. The other frequencies of a given note are approximately the integer multiples of the fundamental. (There is a property called inharmonicity which causes some deviation at high multiples, but I have found it not to be a problem: it only applies at high frequencies which have low amplitudes anyway, and most of its effect as I observed it earlier vanished when I started measuring the mean of each spike, rather than the highest value.) Observations suggest that the relative values of successive spikes at a given time roughly follow an exponential decline; furthermore, this decline is in terms of the frequency, not in terms of the number of the spike. As mentioned before, the spikes themselves are Gaussian, with roughly equal standard deviations. As for decline with respect to time, previous observations have shown how this is unpredictable. In addition, data to approximate this is complicated to assemble because it requires inputs from multiple pianos as well as multiple keys. However, if the string is assumed to be an underdamped harmonic oscillator, then physics suggests that the amplitude over time roughly follows a power law in the general case. Finally, hammer noise should be included in the prior, so that the algorithm is not confused by otherwise unexplained noise. These observations have allowed me to construct a model for how to generate the prior, but as it is untested and relies on arbitrary constants, it adds no new information to the above.

The approach advocated here is complicated in that it takes one difficult problem and replaces it with two slightly less difficult problems. Nevertheless, if implemented and shown to work, this approach could have multiple advantages over other envisioned transcription methods. It uses minimal training data; it identifies how loudly notes are played, relative to one another; and it collects data on the instrument being analyzed. (For example, it could be used to make a synthesizer imitating a recorded piano.) Most importantly, it addresses the fundamental problem behind errors of octaves and similar intervals which plague other approaches.

Although I was ultimately unable to make productive use of the paired music data I collected for this project, I would like to acknowledge and thank the contributors to the Allegro and Audacity open source projects, whose code for handling MIDI files allowed me to collect it.

Of course, special thanks are due to Graham Poliner and Daniel Ellis for laying down groundwork for this research.