CS229 Project: Segmenting Music into Notes using Mixture of Gaussians

Roger Grosse

December 16, 2005

1 Introduction

When we listen to music, we can distinguish individual instruments with relative ease. If a learning algorithm could do this automatically, it would have several applications: computers could transcribe music for us, we could take a recording and edit out the voice for karaoke, we could see what a given piece might have sounded like if Bach had picked one note instead of another one, and so on. A first step towards creating any such system would be to segment a stream of music into its component notes. This remains an unsolved problem, and was the focus of my efforts for this project.

What information would a learning algorithm have to work with? Fig. 1 and 2 show a short excerpt from Hindemith's 'Symphonic Metamorphosis'. The raw .wav input is expressed as air pressure over time (Fig. a), but using the Fourier transform, we convert this to a spectrogram, which gives the amplitude of a given frequency at a given time (Fig. b). Although the former suffices for multiple-source sound separation, the latter is necessary for single-source separation.

What simplifies analysis of music as compared to speech is that there are two basic kinds of sounds: harmonic and percussive. Harmonic sounds, such as those produced by wind and string instruments, consist of a fundamental frequency, which we perceive as the pitch, and a set of overtones, which all have frequencies that are integer multiples of the fundamental frequency. I will refer to these as the spectrum of the sound. We humans use all of these overtones, not just the fundamental, to perceive the pitch and the timbre. A key factor simplifying the analysis of music is that these sounds have a more or less steady pitch over time. Percussive sounds, which I have not tried to deal with yet, have a much more complex spectrum than do harmonic sounds.

In this paper, I will describe the non-negative matrix factorization algorithm, which has been used to separate music into different components. I will show how NMF is not able to separate out notes well enough for an instrument separation system, Finally, I will outline a mixture of Gaussians model I developed to fix the shortcomings of NMF, and then compare the advantages and drawbacks to each approach.

2 Non-negative Matrix Factorization

My first attempt at segmenting music was the non-negative matrix factorization algorithm used by Helen and Virtanen [1]. (They used this as part of a larger system to separate music into its harmonic and percussive parts.) The function of NMF is to separate the sound into different components, each of constant spectrum and time-varying amplitude. In theory, each of these components should roughly correspond to one pitch played by one instrument.

We start with a spectrogram S. (I do not discuss how to get this, because it is a standard procedure, and Matlab does it automatically.) We take the norm of each entry to get the amplitude spectrogram A. We then use the non-negative matrix factorization technique presented by [2] and [1] to break the signal into its components. Although this is not how NMF is usually presented, it will be convenient for this paper to formulate the algorithm in terms of maximum likelihood estimation. We pretend that there is some underlying distribution $\mathfrak{D}$ over time and frequency, and that the amplitude spectrogram A is a weighted dataset drawn from $\mathfrak{D}$. We want to find the distribution $\mathfrak{D}'$ which assigns maximum likelihood to A, subject to the following constraints:

1. First, a component c is selected according to some distribution P over C different components.

2. After the component is selected, a point x is drawn from a joint conditional distribution $p(x|c)$ over time and frequency. The time and frequency are independent given the component, e.g. $p(x|c) = p_f(x|c)p_t(x|c)$.

In other words, each component has constant spectrum and time-varying amplitude. To find $\mathfrak{D}'$, we use an iterative NMF algorithm described by Lee [3] to estimate

$$A \approx WH, \text{ where } W \in \mathbb{R}^{m \times r}, H \in \mathbb{R}^{r \times n}$$

using the KL divergence cost function. The columns of W represent the marginal distribution over frequency for each component, and the rows of H represent the marginal distribution over time. Each product $W_i H_i$ of a column of W and the corresponding row of H represents the joint distribution for one of the components.

We used Matlab NMF code provided by Hoyer [4].

Once we have expressed the joint distribution in terms of the components, we resynthesize each component. Helen and Virtanen do not mention how they do this, and I don't know if there is a standard method, but I tried several approaches and arrived at the following: to synthesize component c, take the original spectrogram S (not A, because we need the phase information). Apply a time-varying filter where, at each time step t, the amplitude for frequency f is multiplied by $p(c|f,t)$ according to our model. Use the inverse Fourier transform to convert this back to a signal in the time domain. This approach has the convenient property that, when the component signals are summed together, the result is the original signal.

NMF was generally able to separate out individual notes in toy examples consisting of fewer than 10 notes. However, when I ran it on selections of real music consisting of 20-30 different notes, there were several major problems that made it unrealistic to cluster the components. Rather than separate out different notes, the algorithm would separate out a note's overtones as different components. More significantly, some overtones overlapped between multiple pitches, and these would be traced as one continuous component. I could not see any way to use these components as input to a clustering system without serious additional processing. Additionally, in these real examples, the separated components would be highly noncontinuous over time, resulting in static.

(I think this is because real instruments tend to fluctuate slightly in pitch, and NMF does not recognize continuity of pitch.) I could make this sound more pleasant by blurring the rows of H, but I regard this as a hack, and it would be much more satisfying if the segmentation algorithm itself could enforce continuity over time.

The problem with NMF is that it is underconstrained: we could permute all the rows and columns of X and wind up with the same (permuted) factorization. The algorithm does not take into account either the pattern of overtones or continuity over time.

3 Mixture of Gaussians

To enforce both overtones and continuity over time, we add the following two constraints to the NMF model:

1. The conditional distribution over time $p_t(x|c)$ is a Gaussian with mean μ_t^c and variance σ_t^{c2}.

2. The conditional distribution over frequency $p_f(x|c)$ is what is called a tied mixture of Gaussians. The two parameters of this distribution are the fundamental frequency f_0^c, and the variance σ_f^{c2}. The distribution is the normalized sum of the n Gaussians with mean kf_0^c and variance σ_f^{c2}, for $k = 1, \ldots, n$, where n, the number of overtones, is a fixed in our model to be 6. (Note: after I came up with this mixture of Gaussians model, I did a Google search and discovered that Kameoka et al. [5] had already used a tied mixture of Gaussian model for pitch estimation at a single time, and derived EM updates for f_0 and P similar to the ones I describe below. The rest of this model, including the modifications to aid convergence, has not been done before, to my knowledge.)

This model makes some unrealistic assumptions, which I discuss in Analysis. We can maximize this using the EM update rules given in the appendix.

3.1 Modifications to aid search

There is one necessary adjustment to the above algorithm before it can detect any notes. In general, a very small percentage of the area of a spectrogram is taken up by notes (the red parts of Figure b). The rest of it (the blue area) consists of very quiet noise. Although the amplitude of these blue areas is orders of magnitude softer than the amplitude at a (t, f) point corresponding to a note, when taken together, they constitute a large probability mass. Because the noise is attributed to the components, this results in unacceptably large variance in both frequency and time for each of the components, preventing them from converging to any notes at all. To solve this problem, it is necessary to postulate an additional noise component, which is held fixed as a uniform distribution over all times and frequencies. The result of this is that, in the E-step, almost all of the blue areas are attributed to this noise component.

Once the noise component is factored in, the algorithm will generally converge to a local optimum which includes about 3 or 4 notes in a selection of about 20. The components which do not converge to notes do one of two things: they converge to a cluster over multiple notes, or they gradually diffuse and die away to zero probability mass. In the second case, we randomly reinitialize the parameters of the dead component and assign it large probability mass. Often, these random restarts will then converge to notes. When run for 200 iterations, the algorithm will often find 8 to 10 notes, depending on the difficulty of the selection.

Because efficiency is a big concern, I used 500ms windows for this algorithm rather than 40ms. Although I would eventually like to use smaller window sizes, for the moment, I believe it is far more important to confront the issues I discuss below.

4 Analysis

Because I don't know of any standard evaluation metrics for instrument separation systems, I did not try to compare this system quantitatively against NMF. However, I will present some of my qualitative findings in comparing the two systems, show the advantages and disadvantages of each approach, and outline some additional changes I would like to make to the mixture of Gaussians model. Some examples of system output are shown in the appendix.

The appeal of NMF lies in that it is easy to implement (under 10 lines of Matlab code) and fast (roughly 10 minutes for 10 seconds of music). The articles I found did not need to use any special tricks to make the algorithm work for music separation. Furthermore, it is versatile: it fits both harmonic and percussive music under the same framework, and Helen and Virtanen were able to use it to separate the harmonic and percussive components of music (though the separation was far from clean). Unfortunately, as I mention above, the algorithm does not produce anything that even roughly corresponds to notes, and would not be usable as part of an instrument separation system without serious additional processing.

The mixture of Gaussians model, by contrast, actually separates out real individual notes. A further benefit is that you do not need to know ahead of time how many notes are present in the music; an overestimate is OK. The extra components simply do not converge. Unfortunately, the EM algorithm only finds a subset of the notes. Here, I outline some of the common errors:

1. Low recall. Generally, the algorithm quickly finds the longest/most prominent notes in the selection, and finds the medium-prominence notes after hundreds of iterations, but does not generally pick up minor notes (say, eighth notes). It could just be that it is a hard problem to determine what sounds are really notes and what sounds are background noise, but I think it's still possible to capture these in the existing framework.

2. Clusters of notes. Often, a component will converge on a cluster of notes, rather than on a single note, as in the yellow area in the figure. This case is usually numerically obvious, because these components have a large variance over frequency. A simple solution would be to automatically split up components that converge to something with high variance; I think this will work, but I haven't gotten around to implementing it.

3. Fundamental frequency errors. Local optima are an issue for the tied Gaussian model, because a component will sometimes converge on an f_0 which is either half or twice the true f_0. This can probably be solved by periodically checking whether the data likelihood can be increased by halving or doubling the fundamental frequency.

What about the assumptions of the model? I believe the tied Gaussian model is approximately realistic for spectra of harmonic instruments. However, the overtones are not always exact integer multiples of the fundamental frequency, and in some cases, this causes the algorithm to miss some overtones. A simple solution would be to add a slack term to the mean of each overtone, e.g. instead of requiring the mean to be kf_0^c, let it be $kf_0^c+\epsilon$, where ϵ is a normally distributed error term.

The more dubious assumption is requiring the distribution over time to be Gaussian. Musical notes are not even approximately Gaussian. In practice, this is not a problem for instruments with a continuous sound, such as winds and strings, because the resynthesis stage only needs to know what percentage of a given time/frequency pair to attribute to a given component. If only one instrument is playing at a given frequency at a given time, which is usually the case, the entire note will be attributed to that one component, as wanted. However, the Gaussian assumption is a serious drawback for instruments with a percussive impact, such as piano or guitar. The amplitude at the initial onset of the note is an order of magnitude larger than that of the rest of the note, causing the algorithm to attribute multiple Gaussians to one note. Handling these instruments correctly would require significant changes to the basic model.

The other serious drawback to this system is that it is unreasonably slow: it currently takes about 30-60 minutes per 100 iterations, and the algorithm continues to find new notes after hundreds of iterations. Although this is an order of magnitude slower than NMF, it is not reasonable to compare the two, because I feel this algorithm does a deeper kind of computation than NMF, in that it identifies actual notes. Nevertheless, it is currently too slow to be usable for any large selection of music. Thus, it would be useful to find ways to reduce the number of iterations required for the search, or approximate update rules that shorten the time required for one iteration.

Clearly, I have not solved the problem of music segmentation, but I think the results show that this algorithm finds enough interesting clusterings that, with more modifications, it could probably achieve good performance. I would like to continue to improve this algorithm to the point where it can be used as a component in an instrument separation system.

5 Collaboration

Special thanks to Gunjit Singh.

6 Appendix: EM Update Rules

E-step

$$\begin{aligned} Q_{f,t}(c,k) &= p(c,k|f,t) \\ &\alpha \quad p(f|c,k)p(t|c)p(k|c)p(c) \\ &\alpha \quad p(f|c,k)p(t|c)p(c) \end{aligned}$$

where the $p(k|c)$ terms drop out because they are assumed to be $\frac{1}{n}$ for any c, k.

M-step

$$\begin{aligned} P(c) \quad &\alpha \quad \sum_{f,t}\sum_{k=1}^{n} w_{f,t}Q_{f,t}(c,k) \\ \mu_t^c &= \frac{\sum_{f,t}\sum_{k=1}^{n} t w_{f,t}Q_{f,t}(c,k)}{\sum_{f,t}\sum_{k=1}^{n} w_{f,t}Q_{f,t}(c,k)} \\ \sigma_t^c &= \frac{\sum_{f,t}\sum_{k=1}^{n} (t-\mu_t^c)^2 w_{f,t}Q_{f,t}(c,k)}{\sum_{f,t}\sum_{k=1}^{n} w_{f,t}Q_{f,t}(c,k)} \\ f_0^c &= \frac{\sum_{f,t}\sum_{k=1}^{n} \frac{f}{k} w_{f,t}Q_{f,t}(c,k)}{\sum_{f,t}\sum_{k=1}^{n} w_{f,t}Q_{f,t}(c,k)} \\ \sigma_f^c &= \frac{\sum_{f,t}\sum_{k=1}^{n} (f-kf_0^c)^2 w_{f,t}Q_{f,t}(c,k)}{\sum_{f,t}\sum_{k=1}^{n} w_{f,t}Q_{f,t}(c,k)} \end{aligned}$$

References

[1] Helen, M., Virtanen, T. "Separation of Drums from Polyphonic Music using Non-Negative Matrix Factorization and Support Vector Machines." in proc. 13th European Signal Processing Conference Antalaya, Turkey, 2005

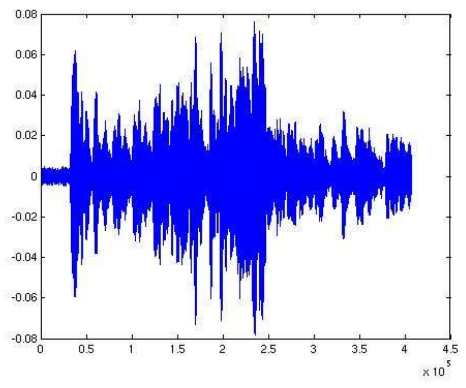
[2] Smaragdis, P., Brown, J. "Non-negative Matrix Factorization for Polyphonic Music Transcription." in IEEE Workshop on Applications of Signal Processing to Audio and Acoustics, p. 177-180, (2003).

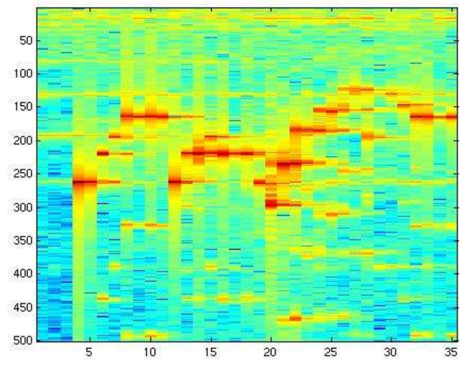
[3] Lee, D., Seung, H. "Algorithms for Non-negative Matrix Factorization." in Neural Information Processing Systems (2000).

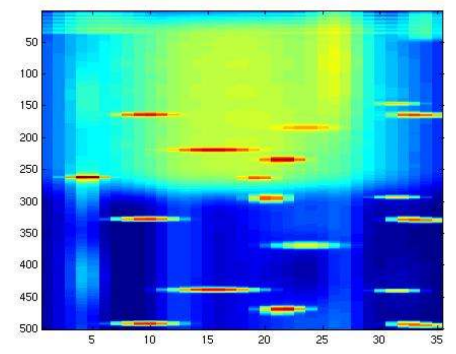
[4] Hoyer, P. "Non-negative Matrix Factorization with Sparseness Constraints." Submitted. Available online: http://www.cs.helsinki.fi/patrick.hoyer/

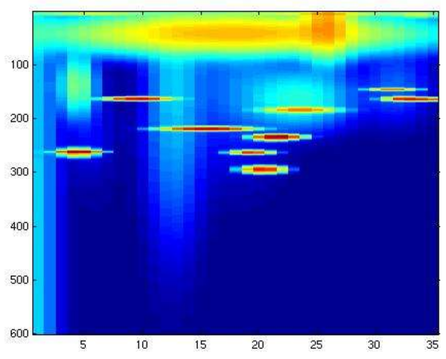
[5] Kameoka, H., Nishimoto, T., Sagayama, S. ”Separation of Harmonic Structures Based on Tied Gaussian Mixture Model and Information Criterion for Concurrent Sounds.” in IEEE International Conference on Acoustics, Speech, and Signal Processing, Montreal, 2004.

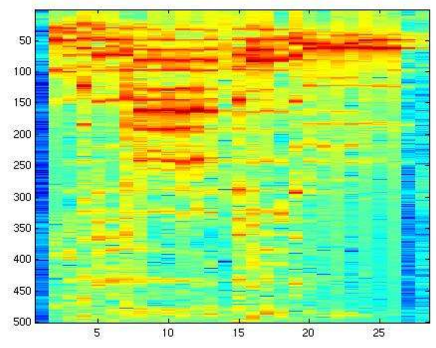
Appendix: figures

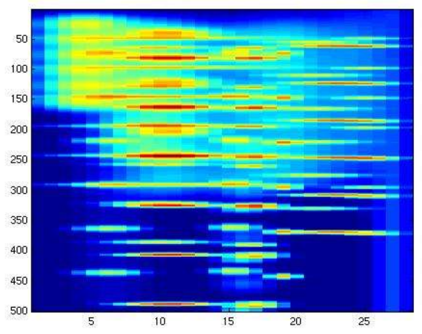

(a)

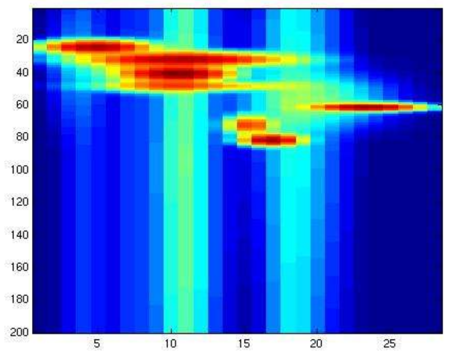

(b)


(c)


(d)


(e)


(f)


(g)

(a) Wave representation of an excerpt from Hindemith's “Symphonic Metamorphosis.”

(b) Spectrogram of this same excerpt. Vertical axis is frequency, and one unit represents 5.38 Hz. Horizontal axis is time, one unit represents 0.25 seconds. This is a relatively easy excerpt because it is monophonic, consisting of just one flute.

(c) System output for this excerpt. The algorithm found 9 notes. The red areas represent the different overtones of a note on which the algorithm converged. Green, orange, and light blue areas represent components which have not yet converged. In a full working system, only those 9 notes would be reported.

(d) Same as (c), but only the fundamental frequencies, and not the overtones, are shown.

(e) Spectrogram of an excerpt from Billy Joel's “And So It Goes.” This is a harder example, because it is polyphonic, and because it consists of human voices.

(f) System output. The algorithm found 7 notes.

(g) Same as (f), but only fundamental frequencies are shown. Note that it is zoomed in vertically.