Hidden Markov Models and Automatic Music Transcription

Brendt Gerics

April 20, 2014

Abstract

Hidden Markov models (HMMs) are fairly new in the field of stochastic studies, and play a key role in the advancement of machine learning and inference techniqes. In this paper, I offer a framework that uses a HMM to make software that will automatically typeset a song that someone plays by breaking a song down into discrete observable events, and using an HMM to extrapolate the desired information.

When we look at a traditional Markov chain, we make the assumption that the states we are interested in can be completely determined by merely observing them. That is, the current state is known at any time. A good example of this is games or sports. At each transition, there is no question as to what is changing in the game. In real life applications however, we cannot always determine the state of a system through observation. We must find some other way to uncover the underlying states. What we often see instead is some type of emission from a state that corresponds probabilistically to what that state may be. The fundamental idea of hidden markov models is that we can make some mathematically sound guesses about the hidden sequence of states, provided that the hidden states are a stochastic process.

There are two general situations that a hidden Markov model can address. The first is when we want to know about some sequence of states that has already occured, but of which no direct record was kept, but rather some other set of observations relating to those states. Here, the states are hidden because the past cannot be observed.

The other case in which HMMs can be used, is when the states in question are active, but no information can be directly observed. A good analogy is diagnosing illnnes–the states are there and unknown, but observations can be made (blood tests, temperature, ...) the relate probabilistically to the person's health.This paper will fall into the former category.

Let me start by describing the notation I use in this paper, first musical, then mathematical. The words note and pitch are interchangeable in this text,and both are subjective terms. Pitch is related to frequency; frequency represents the rate of vibrations in the air of some specific sample of sound, and pitch (or note) refers to a consistent sound who's frequency is "close" to one of finite pre-determined frequencies. This paper is based on the western definition of pitch, which is best conceptualized by a piano keyboard. First we define the colored note on the attached sheet as A_4, and assign a frequency of 440 Hz. This is the most common definition used. Then the rest of the white notes are then labeled with the letters as shown.

Now I define the notion of sharp and flat. $A_3^\sharp$, pronounced A three sharp, refers to the note immediately higher to A_3 on the keyboard. Likewise, $G_3^\flat$, pronounced G three flat, refers to the note immediately below G_3. Notice that these are both the same note on the keyboard. This fact is not really important to the process involved, since it is only a matter of labeling. [1] Finally, a rest is just a moment of silence in music, in which there is no sound being played. Now the frequencies assigned to each note is defined in relation to our initial note A_4:

$$f_n := 440 \cdot 2^{\frac{n}{12}}$$

Here n represents how many steps away from A_4 we want to go, positive n

[1] Also note that $E^\sharp$ is identical to F and $F^\flat$ is identical to E, but this is rarely used this way in music.

indicating higher notes, and negative n indicating lower notes.

Now I define the basic components of HMMs [2],[3]:

$$\begin{aligned}
Q &= \{q_1, q_2, \ldots, q_N\} \quad \text{Set of } N \text{ distinct states.} \\
V &= \{o_1, o_2, \ldots, o_M\} \quad \text{Set of } M \text{ distinct observations} \\
A &= \{a_{ij}\} \quad \text{The transition matrix from state } i \text{ to state } j. \\
B &= \{b_j(k)\} \quad \text{The emissions matrix; } b_j(k) = P(o_k|q_j) \\
\pi_i &= \text{initial distribution} \\
O &= (O_1, O_2, \ldots, O_T) \quad \text{Observation sequence, where } O_i \in V \\
S &= (s_1, s_2, \ldots, s_T) \quad \text{Sequence of specific states}
\end{aligned}$$

The only phrase that may need explaining here is $b_j(k) = P(o_k|q_j)$. In words, this is the probability that we will observe an observation given some state, e.g. the probability of seeing a fever, given that the patient is sick. The literature on Hidden Markov Models is fairly well established, and nearly all will mention the same three ~~problems~~ motives[2]:

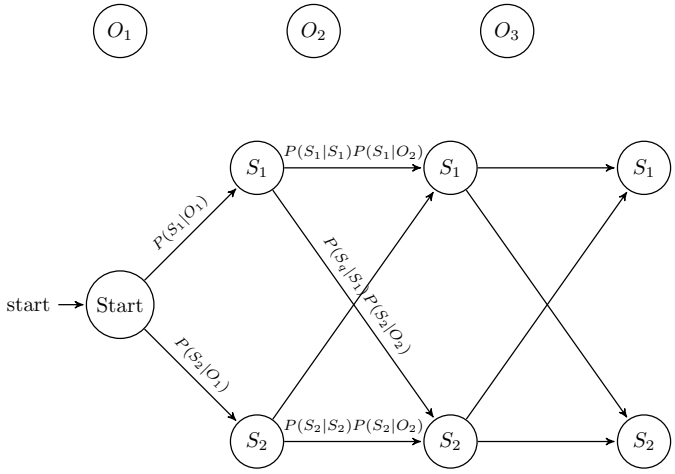
1. Evalution – Given an observation sequence O and a model λ, what is the likelihood of the model producing that sequence? Mathematically, how can we determine $P(O|\lambda)$?

2. "Best" path – How can we determine which sequence of states would have most likely produced some given observations sequence? Here we want to uncover what is hidden.

3. Training – If we start with an observation sequence, how can we adjust the parameters of λ to fit the data better?

Motive 1 Let $\lambda = \{A, B, \pi\}$ be a model, O a sequence of observations, and let S denote a state sequence $(s_1, s_2, \ldots, s_T)$. The most straightforward way to calculate the probability of O is to sum the probabilities of every possible state sequence. That is to find $P(O|\lambda) = \sum_S P(O, S|\lambda)$. This is where what is known as a trellis diagram can be useful:

[2]I do not care for the word problem – it is too pessimistic to label mathematical inquiries as problems. Motive seems to indicate a more authentic look at math



This gives a graphical representation of what summing over sequences of states looks like. Moving to the right represents moving through time via our observations. Each time we move right, we are "choosing" a path to calculate. It is important to note that for any particular observation sequence, we either go through state 1 or state 2.[3] Written mathematically, the probability of the observation for a particular sequence of states, S, is given by:

$$P(O|S,\lambda) = b_{s_1}(O_1) \cdot b_{s_2}(O_2) \cdots b_{s_T}(O_T)$$

Further, the probability of a sequence S is given by

$$P(S|\lambda) = \pi_{s_1} a_{s_1 s_2} a_{s_2 s_3} \cdots a_{s_{T-1} s_T}$$

It can be shown that the joint probability of an both S and O occuring simultaneously is the product of these previous two expressions. That is,

$$\sum_S P(O,S|\lambda) = \sum_S \pi_{s_1} b_{s_1}(O_1) a_{s_1 s_2} b_{s_2}(O_2) a_{s_2 s_3} \ldots a_{s_{T-1} s_T} b_{s_T}(O_T)$$

This approach is unfeasible for most real life applications because the of the number of calculations required. For N possible states, with T observations steps, the number of calculations required is just shy of $2TN^T$. As an example, the model I use for the music typesetting has at least 50 observations steps with 28 states. An better approach is necassary, and the answer lies in what is known as the forward variable. It is defined as follows:

$$\alpha_t(i) = P(O_1 O_2 \ldots O_t, s_t = q_i|\lambda)$$

Here t indicates how far in the observation sequence we are, and the $1 \leq i \leq N$ pertains to the state. α represents the the total probability of all paths that

[3] Or in general, we can only go through one of N states.

pass through that particular state as of time t. A trellis diagram is often used to show this, and for good reason. The algorithm to find $\alpha_t(i)$ is known as the forward algorithm, and has three steps:

1. Intitialization This is pretty silly. The first step is just calculating $P(q_i|O_0)$ for each state. See the diagram.

2. Induction The rest of the α 's can be found by multiplying the previous values of α's by the paths leading to the state in question. I know what I'm saying, I can't figure out how to word it yet. Mathematically written:

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i)a_{ij}\right] b_j(O_{t+1}), \quad 1 \leq t \leq T-1,\ 1 \leq j \leq N$$

3. Termination: When we get to the end, we have calculated $\alpha_T(i)$ for all states, and we can get the probability we originally wanted by summing $\alpha_T(i)$ over all states. That is, $P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i)$. This is computationally reasonable, roughly N^2T computations.

We can repeat this process going backwards in what is known as the backwards algorithm. This is identical to the forward algorithm, except we dtart from the end, and perform the calculations until we terminate at the beginning. This is not necessary in calculating the probability of an observation sequence (though it can be used) but is indespensible to the third motive.

Motive 2 To find the most likely state we can take one of two apporaches. The first is to maximize each step individually. This approach is not very reasonable in most cases because it is local, i.e. based only on adjacent states, and does not take into account the entire sequence. Since this is used only in the most trivial models, I will omit the work in this paper. A much more robust approach, which I will use, is known as the Viterbi algorithm. Its purpose is to find the single state sequence that maximizes a given sequence of observations. To do this we define the following:

$$\delta_t(i) = \max_{s_1 s_2, \ldots s_{t-1}} P(s_1 s_2, \ldots s_t = i, O_1 O_2 \ldots O_t|\lambda)$$

The Viterbi algorithm is structurally very similar to the forward algorithm. Some differences are that here we keep a counter on the best path, and rather that summing over paths, we are maximizing over paths. Here, our counter is $\phi(i)$ [3], [1].

1. Initialization:

$$\delta_1(i) = \pi_i b_i(O_1), \quad 1 \leq i \leq N$$

$$\phi_1(i) = 0$$

2. Recursion:

$$\delta_t(j) = \max_{1 \le i \le N} (\delta_{t-1}(i) a_{ij}) b_j(O_t)$$

$$\phi_t(j) = \arg \max_{1 \le i \le N} (\delta_{t-1}(i) a_{ij})$$

3. Termination

$$P^* = \max_{1 \le i \le N} \delta_T(i)$$

$$s_T^* = \arg \max_{1 \le i \le N} \delta_T(i)$$

4. Backtracking: Now work backwards from s^* and choose the best states:

$$s_t^* = \phi_{t+1}(s_{t+1}^*)$$

Motive 3 The most important question we can ask, is how do we modify a model to fit the data better. This is important for times that our model is just an estimate from the available information, and we seek to find an explanation that arises from evidence rather than guess work. This process is known as the Baum Welch Algorithm, which is a special case of the well known Expectation Maximization algorithm. The general idea is that given a sequence of observations, we look at the number of expected transitions to each state according to what the model predicts and then compare it to the number of transitions expected from what the observation sequence suggests. From here, new parameters can be determined, giving a new model. This process is then repeated again. With adequate steps, the model will converge to a specific model that reflects the given observation sequence. With the model I created for this project, the Baum Welch algorithm was superfluous, and the math goes beyond the scope of what I set out to learn. If the complexity of this project were to be increased, it would be necassary to invoke this algorithm.

Attempting to model music is difficult because music varies so much from piece to piece. It is comparable to describing the weather. It may be easy to recognize the different weather patterns (music types), but giving an explanation to the mechanics underneath is difficult. I was guided by two key ideas. My first idea was to take a categorical approach by forming the model around like sounding songs. For this project, I decided to narrow in on ragtime music via the work of Scott Joplin for two reasons. First, Scott Joplin's sheet music is readily available on the internet, and secondly it is fairly "regular", i.e. it is easy to pick out a song written by Scott Joplin. My next idea was guided by what is (nearly) universal to music world wide, and that is the idea of a musical key. A key is a set of notes based around one chord, which is just a set of 3 or more notes. Deviating from the key leads to musical tension, which can be enjoyable in moderation, though most people have a limit. Pop music stays in

keys strictly, while atonal music may actually make people angry.[4] A key can be formed around any note, so I reasoned that it was best to convert my data into one key. I chose the convient choice of C major, which consists of the white keys on a keyboard.

In order to create a trasnistion matrix, I cataloged sheet music by Scott Joplin and transposed them into the common key of C major, as mentioned above. I then recorded each transition between notes and divided it by the total number of note changes to get the transitions matrix. For each note, I shifted some probability to all entries within 13 notes (one octave in musical terms) that had zero as a value. I did this so that new songs, which could contain unique note patterns, could be accounted for. If I did not give some probability to these options, novel music would not be able to be typeset. I should mention that for simplicity I chose notes in a fairly narrow range, but still broad enough to cover the melodies of the songs I chose. More specifically, I chose E3 through G5 (see attached sheet) .

Before I could form an emissions matrix, I had to determine what my observations would be. For this, I looked to pitch detection algorithms. I opted for what is known as the YIN algorithm, which is a time-domain algorithm. This means that it determines the pitch of a sound sample by looking at how often its peaks occur. The Yin algorithm is a modification of the well known autocorrelation algortihm. Autocorrelation is the process in which a sample of a signal is shifted and compared to itself in by finding the most likely rate of repetition of patterns. The modifications that distinguish the YIN algorithm from the autocorrelation are beyond my current skill level, but the reference is attached. The software I use, Sonic Visualiser, allows me to run the YIN algortihm every 6 milliseconds (the step size can be adjusted). It is very important to note that the YIN algorithm will make a guess regardless of the signal. That is, if there is a moment of silence in a song, a guess is made that is based on residual background noise. This gives me two basic states to consider: each guess either belongs to an intended note (state 1) or was intended to be silence (state 2). This is decieving, because I do not have two states, for if a step did belong to a note, it could then be one of the 27 that I chose for my range. My total number of states is then 28–27 pitches and 1 null reading, which I label N. I then wrote a program in C that looks at the pitch estimate for each interval and assigns to it the note that it lies closest to, using a table like the one that follows:

[4]The Rite of Spring by Stravinsky is a great example of this–it caused riots when it was played because of the dissidence it contains.

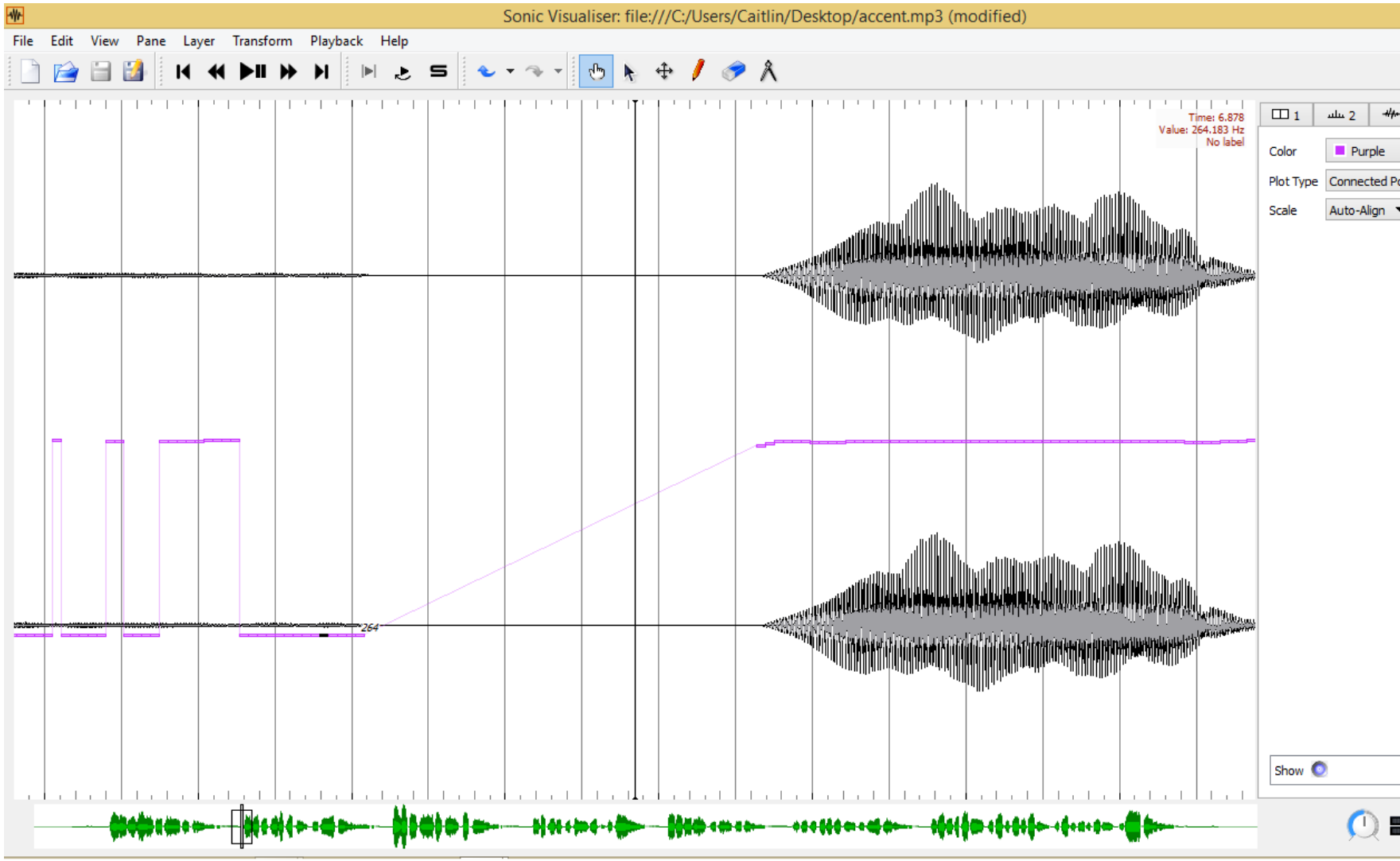
Note	Frequency (Hz)
E3	164.81
F3	174.61
F#3/Gb3	185
G3	196
G#3/Ab3	207.65
A3	220
A#3/Bb3	233.08
B3	246.94
C4	261.63
C#4/Db4	277.18
D4	293.66
D#4/Eb4	311.13
⋮	⋮

If two or more consecutive intervals were assigned the same note, then they were combined into one new interval of longer length. I did this for two reasons. First, the YIN algorithm is accurate enough, that I can assume that if I play a note fairly cleanly, the software should come very close to identifying the pitch. Secondly, and more importantly, it cuts my number of observation steps by a factor of about ten while making my observations discrete. This program also makes the emissions matrix easy to construct. If I play a pitch and get a reading, I am over 95% likely to see that note observed. Any deviations lie only a note on either side. That is, if I play an $A_4^\sharp$, I am most likely going to observe an $A_4^\sharp$, with a small chance of observing an A_4 or B_4. Counter to my inuition, the guesses that the Yin algorithm made during rests were more likely to emit notes not in the key. The initial distribution did not really seem to effect much, though I set it as likely to 90% likely to starting on a note in the key, and 10% likely to start outside of the key. The final page shows what a series of guesses looks like in sonic visualiser. I was sure to include an example of a a ptich-guess that was made during a silence, to show what a false read would look lile.

With my model $\lambda = \{A, B, \pi\}$, what I have is a list of notes, starting times, and durations. I then used Sage to run the Viterbi to determine which notes should be considered, and which should be thrown out (what I labeled N). Once I discarded the unwanted notes, I had Sage output the information as a TEXdocument using the musix TEX package. Rests in music were found by looking at the space between the end of one note and the onset of the note that follows it. The result of my program are attached.

This model was only a proof of concept. With an ideal recording, and under rather simple conditions, I found that my program accurately discarded over 90% of unintended notes, and identified the notes of the melody I played at an even higher success rate. The only problem I had was in calculating rests, which

were only about 60% accurate, though this should be easy to fix, and was most likely the result of insufficient coding.


Sonic Visualiser: file:///C:/Users/Caitlin/Desktop/accent.mp3 (modified)
File
Edit
View
Pane
Layer
Transform
Playback
Help
Time: 6.878
Value: 264.183 Hz
No label
264
1
2
Color
Purple
Plot Type
Connected Po
Scale
Auto-Align
Show

References

[1] Guy Leonard Kouemou (2011). History and Theoretical Basics of Hidden Markov Models, Hidden Markov Models, Theory and Applications, Dr. Przemyslaw Dymarski (Ed.), ISBN: 978-953-307-208-1, InTech, DOI: 10.5772/15205. http://www.intechopen.com/books/hidden-markov-models-theory-and-applications/history-and-theoretical-basics-of-hidden-markov-models

[2] L.R. Rabiner, A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, Vol. 77, No. 2, February 1989, http://www.cs.ucsb.edu/ cs281b/papers/HMMs

[3] Mark Stamp, A revealing Introduction to Hidden Markov models, San Jose State University, September 28, 2012, www.cs.sjsu.edu/faculty/stamp/RUA/HMM.pdf

[4] M. Mauch and S. Dixon, pYIN: A Fundamental Frequency Estimator Using Probabilistic Threshold Distributions, in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2014)*, 2014