

Analyzing Chord Progressions in Popular Music using Techniques from Artificial Intelligence

Jonathan Gordon '07 and Luke Hunsberger
Computer Science Department, Vassar College

Introduction

We're interested in whether computer agents can learn to understand a style of music. The effectiveness of this learning is measured by the computer's ability to predict the next chord in a song it hasn't been shown before.

The accuracy of predicting how music will progress is clearly dependent on the strength of the underlying musical pattern. If the chords in a song are random, then there is no way to predict them, but the song also wouldn't be very listenable [2]. In popular music, there are not only frequently used song structures but also individual chord changes that are more common than others and can be found in songs by many different songwriters. By studying a corpus limited to a single band and a limited range of activity, we hoped to focus on those progressions that are characteristic of its style rather than to music in general.

Having trained on a selection of songs, the system would then have an internal representation of the chord progressions it had seen and how likely each one is. With this, it could make an intelligent guess at how a song would continue, based on the most recently seen chords.

Methods

Relative N-Grams

Representing Songs

One of the key design decisions was how to represent songs. For flexibility and ease of parsing, we stored all the songs as a plain-text series of chords. For instance, the song file for "We Can Work It Out" begins (with the metadata removed): D C D C D G. Here capital letters represent major chords while lowercase letters represent minor chords.

We want the system to look at these chords in terms of their changes, so, for instance, the change from A major to B minor is represented by the vector (2 M m) where the "2" indicates that the root of the chord shifted up two semitones (from A to B), and the "M" and "m" indicate a shift from a major chord to a minor chord. Similarly, the transition from B minor to G major would be represented by the vector (8 m M).

Augmented and diminished chords (represented by a + and an o respectively) present a challenge since, for example, the C+ and E+ chords contain the same notes—namely C, E, and A#—even though they have different roots. (It is this feature which enables augmented and diminished chords to serve as transitional chords when shifting to a new key.) For this reason, when we represent transitions involving augmented or diminished chords, the shifts in semitones are taken modulo four and modulo three respectively—a transition from C to G+ would be represented not by (7 M a) but by (3 M a) since $3 \equiv 7 \pmod{4}$.

N-Grams

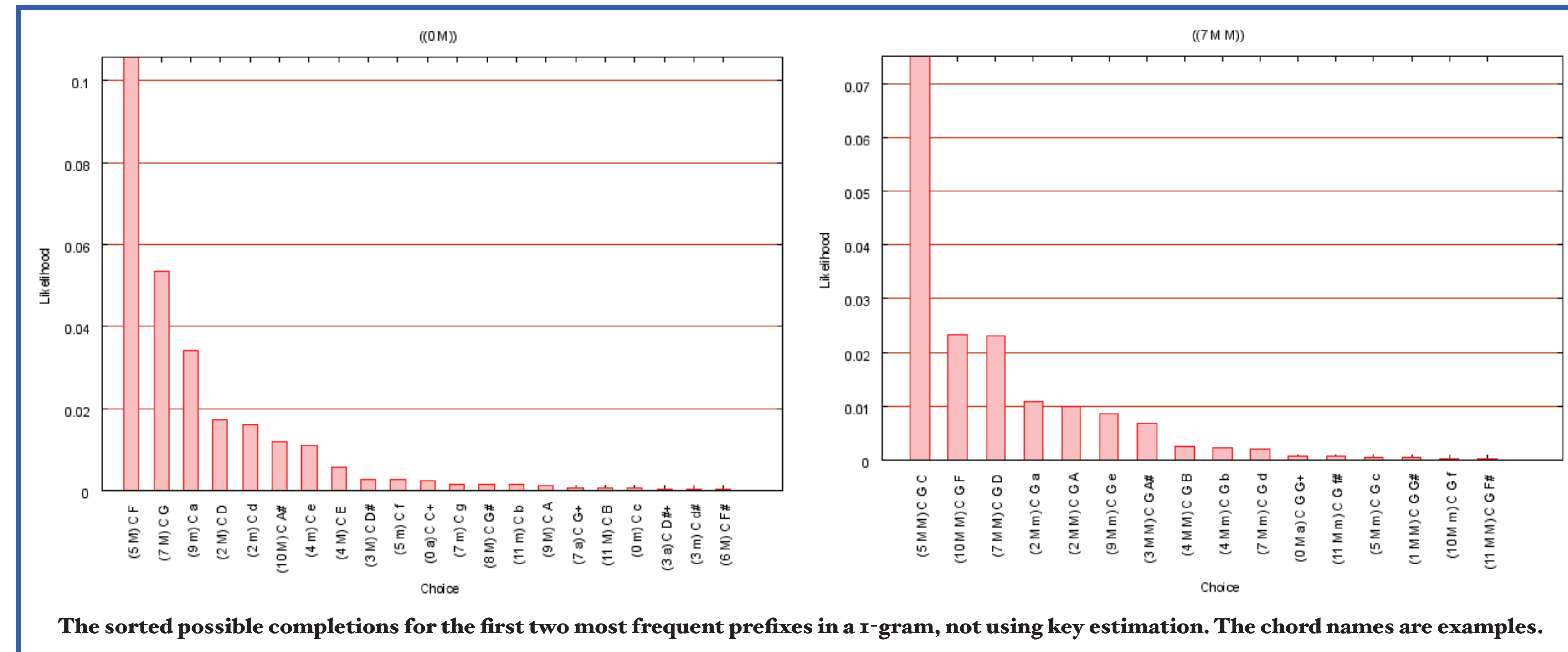
An n -gram model is based on predicting the next item in a sequence given the n most recently seen items in that sequence. For example, a 2-gram model would base its prediction of the next item in a sequence on the previous two elements. In natural language applications, the items in question are words and so a 2-gram model would predict the next word in a sentence based on the two most recently read words. So, if the two most recently seen words were "How are", then a 2-gram model might predict that the most likely next word would be "you", while "forest" would be extremely unlikely.

In a musical domain, one might construct an n -gram model where the data were individual chords, but this model would not be very flexible, nor would it reflect the way humans seem to experience and remember music. Jeff Hawkins points out that the brain uses a pitch-invariant representation of music, which is how we can recognize a melody in any key. And, unless you have perfect pitch you're unlikely to distinguish between the same song in two different keys without hearing them played back to back. This is because the brain instead stores the intervals between the notes [1].

Similarly, our system's "memory" remembers the transitions (i.e., the delta vectors like (4 M m)), not absolute chords (C, F, a). Another advan

Abstract

One of the defining elements of a style of music is its structure, which in popular music is evident in song chord progressions. This project explored the extent to which techniques from artificial intelligence could be used to learn the most likely chord progressions in a corpus of popular songs. N -gram models have been used in natural language processing systems to enable computers to predict the most likely "next word" in a sentence based on the n most recent words. For this project, we built a system based on n -grams to predict the most likely "next chord" in a song given the n most recent chords. In our system, we considered the musical intervals between successive chords, the types of chords (major, minor, augmented, and diminished), and a dynamically computed estimate of the current key of a song. We used a set of 80 early Beatles songs as a test case. The system would be trained on 80% of the songs in the corpus (based on a random split), and we then measured the performance of the system on the remaining 20% of the songs. We found a success rate that increases with the value of n , reaching about 60% accuracy with a 3-gram not using key-estimation. Comparing the accuracy of using key-estimation to not proves tricky due to the nature of the model.



tage of focusing on transitions is that we require exposure to far fewer songs in the learning phase since, having seen the sequence C, F, A#, we can now recognize the possibility of the sequence D, G, C, and many others. Based on a history of one transition (this example would be for a 1-gram) (D to G), we would predict the next transition (G to C). We can do n -grams for any n , with higher values meaning we have more of a history before we make a prediction [2]. Increasing the length of each n -gram, of course, vastly increases the number of potential sequences that need to be considered for us to make a prediction.

Key-Estimated N-grams

The model described above does not take into account the key of the song being examined. So, for example, a transition from D major to B minor is considered equally likely in two songs that might be in completely different keys. To test the hypothesis that the current key of a song affects the likelihoods of various chord transitions, we built an alternative model in which the current key of a song was dynamically estimated and folded into the n -gram model. We also expected that this would enable the model to deal better with songs involving key shifts, which are quite common in the Beatles corpus.

Representing Chords Relative to the Current Key

Suppose our current estimate of the key is C major and the chords being considered are C, D, and f. Relative to the key of C major, these chords are (0 M), (2 M), and (5 m) since, for example, D is a major chord two semitones above the key of C and f is a minor chord five semitones above the key of C. Since this approach already makes the representation of the chords we store key independent, there's no need to look at them as deltas.

Estimating the Current Key

This approach requires that we can accurately estimate the current key of a song. It's not possible to simply store this information in the corpus since many songs have ambiguous keys or contain dramatic shifts in key, so we must allow our idea of the current key to change during the course of the song.

In our system, we compute the current key based on the notes in the most recently seen chords and the degree to which those notes fit each possible key. For example, if the previous chords were C, F, and G (containing the notes C-E-G, F-A-C, and G-B-D), this would fit quite well with the key of C major; however, it would fit rather poorly with the key of E major, which contains a lot of sharps. The notes that are included in the key are given positive weights, with those that are more likely, such as the root, being weighted more strongly. Notes that would be incongruous with the key are given negative weights, with those that are least

likely being the most negative.

To determine the current key we sum the weights for all of the notes included in the chords we're looking at. The chords are looked at in reverse order, so the most recent is looked at first. The weights computed for the chords are linearly sloped off, so that this most recent chord counts for its full computed weight and those longer ago count less and less. If we use the possible (simple) set of weights 5, -7, 5, -7, 5, 5, -7, 5, -7, 5, where the first 5 corresponds to the root and the subsequent weights to each successive semitone, we see the key estimation for the chords C, F, and G proceed as follows:

Checking G 7 M for key C 0 :
 Note G 7 position 0 has weight 5
 Note B 11 position 4 has weight 5
 Note D 2 position 7 has weight 5
 Score of 15 with slope weight of 1: 15

It goes likewise for F and C, yielding a total weight of 32. We then shift the weights one position to the right (wrapping around) to check the next key (C#). The key that gives the highest weight is chosen as the most likely—in this case, C.

Prediction in this key-based model occurs as with any other n -gram model, except that everytime we look at chords to make a prediction, we must first estimate the key so that we know what key-relative sequence we're looking to complete.

Experiment

To experimentally measure the accuracy of our approaches, we ran a series of runs where we took a corpus of 80 Beatles songs and randomly split them into two groups: songs to train on in the learning phase (80%) and songs to measure prediction on in the testing phase (20%). We did this for n values from one to three. During development, we also tried 4-grams, which yielded improved accuracy, as expected, but do to the prohibitive time of runs using $n > 4$, they are not included in the experiments. We did runs on the same splits using our relative model and using key-estimation.

Additional experiments performed in the course of refining the system included running "backward prediction", which is the same as the normal relative model except that you go through the song predicting toward the beginning to see if the chords that will be played after a given chord are a better signifier of what it is than the chords that precede the chord in question. It was found to be nearly the same. We also experimented with different sets of weights for the key estimation, though an experimental determination of the ideal values is yet to be done (see "Future Work").

When looking at the accuracy of the system to predict the completion of a song, we look at the percent of the time that the next chord is our most likely choice and the percent of the time that it's the next most likely and so on. (However, when interactively predicting, our choices are nondeterministic: The system actually computes the probability of each of its possible completions and then randomly chooses one, with the likelihood of choosing each corresponding to its frequency in the corpus. So if there are three known completions to a sequence, all with nearly the same likelihood, they are all nearly as likely to be given as the completion. If, instead, one choice is much more likely, then it will be much more likely to be what we predict.)

References

- Hawkins, J., with Blakeslee, S. *On Intelligence*. Holt and Co., New York, 2005.
- Thom, Belinda. "Predicting Chordal Transitions in Jazz: The Good, the Bad, and the Ugly". IJCAI-95, Music and AI Workshop Paper, 1995.

Results

The experimental runs showed that larger values of n consistently yielded better accuracy. With the relative model and with the key-estimation model, we are able to achieve encouraging accuracy even with small (< 4) values of n . It is ambiguous whether using key-estimation produces better results than not since there is no 1-to-1 correspondence between the models. A 1-gram in the relative model describes a history of one transition, or two chords, whereas a 1-gram with key-estimation means having a history of exactly 1 chord. So, unsurprisingly if we compare 1-grams to 1-grams, the relative model is more successful, but if we compare key-estimation 2-grams to relative 1-grams, key-estimation does better. Using key estimation is markedly slower due to the additional processing that needs to be done at all stages.

	Relative		Key Estimation	
	1st Choice	2nd Choice	1st Choice	2nd Choice
1 gram	32.6	16.3	34.3	18.6
2 gram	45	19.3	39.6	20.6
3 gram	59	56	47.6	23

This table represents the frequency with which the correct next chord for

Conclusions

Regardless of which model is more accurate, both proved successful, being able to predict the next chord accurately most of the time. The significance of this research lies in part in what it tells us about the applicability of the n -gram technique to music, the effectiveness of using relative encodings for representing chord progressions, and also what it shows about the corpus we used: Although it contains songs that seem quite unusual, the corpus as a whole has a number of characteristic, recognizable progressions.

Future Work

To improve the accuracy of the system, we'd like to incorporate consideration of the duration of each chord, so that a chord that is longer will count more heavily, especially in the key estimation, than one in a series of rapid chord changes. Information about duration is already included in the absolute representation of each song in the corpus.

Another way to improve the usefulness of the system would be to allow it to recognize song structure more explicitly (the n -grams allow for a roundabout model of it). Having the code recognize what the likely ordering of (sub)sequences in a song are would allow better accuracy and also more comprehensible results when running an extended prediction.

One thing that remains to be done in order to improve the accuracy of the results is to experimentally determine the ideal weights for key-estimation. This would be done by comparing the accuracy of results for runs over the corpus using all possible sets of values within a limited deviation from the current ones. Due to the large number of runs this would take and the time for each run to occur, this would require a parallel, distributed run over around 20 computers.

You're Gonna Lose That Girl

9 Mm	5 mm	5 mM	5 MM	4 MM
E	C#	f#	B	E
-	-	4 th	1 st	0 th
				G#
				19 th

Excerpt from the likelihoods results. The top row shows the transition, the middle row shows the actual notes in the song, and the bottom row shows the likelihood of the system predicting that note (which choice it would be) using a 1-gram relative model.