Automatic modeling of musical style

Assayag Gérard (1), Bejerano Gill (2), Dubnov Shlomo (2), Lartillot Olivier (1)
(1) Ircam, Centre Georges-Pompidou
1, place Igor-Stravinsky
75004 Paris
prénom.nom@ircam.fr
(2) Université Ben Gurion

Abstract : In this paper, we describe and compare two methods for unsupervised learning of musical style, both of which perform analyses of musical sequences and then compute a model from which new interpretations / improvisations close to the original's style can be generated. In both cases, an important part of the musical structure is captured, including rhythm, melodic contour, and polyphonic relationships. The first method is a drastic improvement of the Incremental Parsing (IP) method, a method derived from compression theory and proven useful in the musical domain. The second one is an application to music of Prediction Suffix Trees (PST), a learning technique initially developed for statistical modeling of complex sequences with applications in linguistics and biology.

Keywords : Unsupervised learning, Musical style, Compression, Predition Suffix Tree (PST), Probabilistic Finite Automata (PSA), Lempel-Ziv (LZ), Stochastic, Quantization, Constraints, Loop, Redundancy, Musical Parameters, Mar\-kov predictor.

Style Modeling

By Style Modeling, we imply building a computational representation of the musical surface that captures important stylistic features hidden in the way patterns of rhythm , melody, harmony and polyphonic relationships are interleaved and recombined in a redundant fashion. Such a model makes it possible to generate new instances of musical sequences that respect this explicited style. It is therefore an analysis by synthesis scheme, where the closeness of the synthetic to the original may be evaluated and may validate the analysis. Our approach is unsupervised, that is we want an automatic learning process that may be run on huge quantities of musical data. Interesting applications include style characterization tools for the musicologist, generation of stylistic meta-data for intelligent retrieval in musical data bases, convincing music generation for web and game applications, machine improvisation with human performers, computer assisted composition.

The IP method

The Incremental Parsing algorithm is inspired by the analysis part of compression techniques of the Lempel-Ziv family. To understand how an idea derived from the compression field might be useful for our purpose, it is important to see that, as has been stated by several authors, compressing is equivalent to understanding, because in order to encode efficiently incoming information one has to perform a fine analysis of the way redundancy is organized. compression algorithm can be split into two phases: first, it reads the input sequence and constructs a model that captures redundancy, and then it generates the compressed code of this sequence with respect to the model. In our case, the second phase is replaced by a stochastic navigation through the model in order to generate new sequences.

During the generation process, a context-inference scheme is applied. The sequence formed by recently generated objects (a particular suffix of this sequence) is the context, from which a prediction on the next object to come is made with regard to a contextual probability distribution. So, the analysis part must provide a dictionary of such possible contexts along with their possible continuations.

First, the dictionary is empty and IP incrementally reads the sequence. At each cycle, it selects a pattern, from the current position to a further position, such that this pattern is the shortest one which is not already in the dictionary. Every left prefix of this pattern may become a context, and every object that follows this prefix may become a continuation. It is easy to see that an optimal representation (in space) for such a dictionary is a prefix tree where a branch descending from the root to any node is a context, the childs of any nodes are the continuation objects to the context going from root to this node, and where the cardinalities of the subtrees at a certain node encodes the probability distribution for the objects at the the root of each subtree. An optimal representation in generation time is a suffix tree where the context are reversed (i.e. from a leaf to the root) and continuations stored as pointers associated to every node.

We have loosely used the term *sequence* as an ordered list of objects. In order to capture a significant amount of musical substance, we shall, in a pre-analytic phase, cut the musical data (generally in Midi format) into slices which beginning and end are determined by the appearance of new events and the extinction of past events. Every slice has a duration information, and contains a series of channels, each of which contains pitch and velocity information and whatever available musical parameters. These slices are serialized in sequences submitted to analysis (these slices will be referred to by the word *object* or *symbol*, and the set of possible symbols as *alphabet*).

We have described here the basic IP algorithm. This algorithm had already been tested with interesting results, but had certain drawbacks that made it quite impractical in specific situations. The improvements presented here are divided into four sections: pre-analytic simplification, generative constraints, loop escape, analysis-synthesis parameter distribution.

Pre-analytic simplification

Real musical sequences - for example MIDI files of a piece interpreted by a musician - feature fluctuations of note onsets, durations and velocities, inducing a complexity which fools the analysis: the alphabet size tends to grow in an intractable way, leading to unexpected failures, and poor generalisation power. We have thus developed a toolkit containing five simplification filters:

- the *arpeggio filter* vertically aligns notes which are attacked nearly at the same time,
- the *legato filter* removes overlap between successive notes
- the *staccato filter* ignores silence between successive notes,
- the *release filter* vertically aligns note releases,
- the *duration filter* quantizes the durations in order to reduces the duration alphabet.

These features can be tweaked by the user, using thresholds (e.g. a threshold of 50ms separates a struck chord on the piano from an intended arpeggiated chord). Using the simplification toolkit, Midifiles containing real performances that were intractable with the basic IP now become manageable, opening new perspectives, because this particular kind of musical data, full of idiosyncrasy, is of great value as a model for synthetic improvisation.

Generative constraints

It is now possible to specify constraints during the synthetic phase. At each synthetic cycle, if the constraint is not respected by the new generated symbol, this generation is canceled and a new symbol is tried. If no symbol is satisfying, the algorithm backtracks to the previous cycle, and more if necessary. One interesting constraint is called the continuity constraint: at any cycle of the synthetic phase, it is possible that no context sequence be a suffix of the already generated sequence. The maximum context is thus the empty context. In these cases, the algorithm generates a continuation symbol of the empty context, that is to say, any symbol, with a stochastic model corresponding to its occurrence in the original sequence. The obtained result is a musical discontinuity. To avoid this, it is possible to specify a continuity constraint which imposes a minimum size of context anytime during the synthetic phase.

Loop escape

The synthetic phase may easily enter into an infinite loop state. Here is one example: at one cycle, the maximal context *A* proposes only one continuation symbol; once this symbol is generated the new maximal context *B* features only one continuation symbol, and the new maximal context is *A*, again. The next contexts will be *B, A, B, A,* ... This tends to happen when the input data contains contiguous repetitions, which is often the case in music. We describe a mechanism to detect and escape these loops.

The previous example is very simple, because this loop was totally deterministic, and had only two states. At one state of a loop, it may possible to consider several possible continuation symbols, but this choice leads, sooner or later, to return back to this present context or a previous one. We introduce the concept of context-generated subtree, which consists of the exhaustive set of all the possible contexts that may be met after the present one. There is a loop if and only if this context-generated subtree contains few contexts. When the size of the context-generated subtree is below a user-specified threshold *N*, an *N*-order-loop is detected. The loop phenomenon is principally due to the fact that the synthetic phase searches for the maximal context, which is unique and proposes few alternative continuation symbols. In the case of a loop, we loosen this constraint and examine not only the maximal context, but also smaller ones, which escape the loop in most cases.

Analysis-synthesis parameter distribution

The analytic phase consists of finding the redundancy inside an original sequence of symbols. The trouble is there may be so much complexity and diversity in musical sequences that the size of the alphabet may be of the same order of the length of the sequence. Therefore, little redundancy would be observable. Moreover, each symbol consists, as said before, of several channels, each channel consists of notes, and each note consists of different parameters. So a symbol is in fact a Cartesian product of several musical parameters.

In order to increase abstraction and power in the analysis, we allow the system to discard some parameters, for example the velocity. The retained parameters will be called *analysis information*. In this way, it is obviously possible to increase redundancy, because it implicitly organizes the alphabet into equivalence classes: a chord struck two times with different velocities is nevertheless the same chord with the same harmonic function, and it will be detected as such. The problem is, in the synthetic phase, all the discarded information cannot be retrieved. For example, if we choose to discard note durations and dynamics, we finally obtain isorythmic and dynamically flat musical sequences, which sounds like musical box production. A better solution is to store in the model the excluded information.This information is thus called *synthetic information*. The analytic phase is now performed on classes inside the initial symbol set, and during the synthetic phase, synthetic information, e.g. expressivity, may be reconstructed.

This solution has significant advantages. First, generated music regains much diversity, spirit and human appearance of the original one. Moreover, since it is possible to restrict analytic information and therefore find more redundancy in the original sequence, the synthetic phase becomes less constrained. At each synthetic cycle, every context features many more possible continuations than before.

Implementation

The software is implemented as a user library in an Open Source visual programming language developed at Ircam, called *OpenMusic*. Each step of the algorithm - pre-analysis, analysis, synthesis, and post-synthesis, is a function, which, in the musical representation software, is represented by a box featuring inlets and outlets. All parameters may be tuned up by the user. Indeed, synthetic constraints, the distribution of analytic and synthetic information, and the reconstruction of the synthetic information may be explicitly formulated through visual expressions.

Musical experiments

A lot of musical experiments have been carried in order to test the new IP algorithm. Midifiles gathered from several sources, including polyphonic music, piano music, and which style ranges from early music to hard-bop jazz have been submitted to the learning process. Experiments show that the combination of the simplification tool box and the new analysis-synthesis distribution scheme improves dramatically the results in the case of 'live' music, and in cases where the overall complexity leads to a huge alphabet. We show convincing examples, including, a set of piano improvisations in the style of Chick Corea, another one in the style of Chopin Etudes, polyphonic Bach counterpoint, 19th century symphonic music, and modern jazz style improvisations derived from a training set fed by several performers asked to play for the learning system.

PST

Asymptotically it has been shown that IP predictor outperforms a Markov predictor of any fixed finite order. This surprising property of the IP scheme derives from the counting interpretation of the IP procedure. In this interpretation, the IP predictor is viewed as a set of sequential predictors operating on separate bins, where each phrase derived in the process of IP parsing is referred as the bin label. Since IP is unbounded in its length, it can be shown that for any finite k-th order Markov predictor, the long contexts of IP serve as refinements of the k-th order Markov predictors for sufficiently long sequences. Thus, the total number of errors due to a long context prediction is smaller for IP than the error in the case of a limited memory Markov predictor. When calculating the overall error regime, it turns out that asymptotically the longer terms dominate (since the length of the string grows) and eventually the IP scheme outperforms any finite order Markov predictor.

In practice the strings (music sequences) are of a finite order, and moreover, the size of the IP tree is bounded by a small finite size. Due to these limitations it is desirable to consider modeling schemes that might be more optimal for shorter-term situations. Another important feature of IP that seems redundant for our needs is the sequential nature of its operation. In our application, the goal is generation of new sequences that maintain similar statistical properties to the reference source. We use the prediction probabilities as the statistics generator, but we are not bounded by a requirement to rely on the past only. Allowing one to use the whole sequence for estimation of the statistics might help improve the performance.

Ron et al developed a variable length Markov model termed Prediction Suffix Tree (PST). It has been shown that PST is a subclass of Probabilistic Finite Automata called PSAs. PSA is a variant order Markov chain in which the memory is variable and in principle unbounded. Given a finite size PSA, there exists an equivalent PST of a slightly larger size that produces the same probability output. Moreover, an efficient learning algorithm exists that allows one to construct a PST from samples generated by PSA. Now, if we consider the PSA as a common ground relating it to IP, we can consider the similarities and differences between the two approaches. One can see that both methods are similar in terms of their use of a variable length context for determining the probability for next symbol. The basic estimation procedure of the PST though is significantly different from that of IP.

IP/PST comparison

Batch vs. on-line

PST parsing is not on-line in nature, as IP is - the training text is viewed as a whole unit and symbol frequencies are observed over the whole text. Thus there is no *arbitrary parsing* as in LZ where a single symbol change in the sequence may have deep influence on the dictionary structure. So PST may prove more powerful for short sequences but is not practicable for real time improvisation situations, where the on-line nature of IP will be adapted.

Selectivity

while IP goes over the training text and parses all of it, the PST learning algorithm looks at the text as a whole and picks for its dictionary only patterns considered *relevant*. Relevance is defined through several parameters pertaining mainly to the number of times that subsequence has appeared in the whole text, and to how different prediction of the next symbol using that substring differs than that using its suffix (e.g. a PST will learn the suffix/subsequence *BCBA* only if it appears enough times in the text, and the next symbol prediction (*BCBA?*) is different enough than that achieved using only *CBA* (*CBA?*)). This selective parsing can lead to more discontinuities and a smaller repertoire to improvise on, but the context→inference rules stores in the trees may be considered as more motivated than in the case of IP.

From these two remarks, one could say that IP is more adapted in generative applications, especially improvisation and real-time, and PST is more precise and complete, and is a good start for musicology applications where one wants to achieve the finest description.

Other experiments are undergoing and a more precise comparison will appear in the final paper.

Conclusion and new directions

In the present state of researches, we have to consider music as a sequence of symbols. It could be more interesting to analyze directly the bi-dimensional score structure. This would enable an intelligent analysis of a fugue for example.

Although the context-inference approach may be compared with the implication/realisation view of Meyer, we have to acknowledge that his view is more powerful since his expectation is in long term, and not a systemic first order expectation like ours. But we suspect hard computability problems behind the long term approach.

In our work, learning is indeed unsupervised since we do not feed our system with musical knowledge, rather it analyzes music with constrained algorithms and does not proceed to any inductive inference. Another approach would be trying to induce automatically some conclusions about the presence of musical structures, with the help of minimal cognitive mechanisms.

Other statistical techniques have to be experimented in music and compared to the two presented here. PPM for instance, can be thought of as going half-way between IP and PSTs. They are on-line (like IP) but they model *k*-terms of growing *k* with respect to data wealth (like PSTs).

Finally, one may try instead of modeling sequence x1 x2 … xn based on its own redundancies, to try and model x1 x2 … xn based on the redundancies of a second, correlated (eg 2nd voice) sequence y1 y2 … yn (transducer). Then, the generation of a X-type sequence could be constrained by an incoming Y-type sequence. This could lead to a synchronous improvisation scheme, where the computer, instead of providing 'answers' to a human player, as usual, would play synchronously with him, keeping an overall polyphonic consistency.