FOUNDATIONS FOR ON-THE-FLY LEARNING IN THE CHUCK PROGRAMMING LANGUAGE

Rebecca Fiebrink
Princeton University
Computer Science
fiebrink@princeton.edu

Ge Wang
Stanford University
Center for Computer Research in Music and Acoustics
ge@ccrma.stanford.edu

Perry R. Cook
Princeton University
Computer Science (also Music)
prc@cs.princeton.edu

ABSTRACT

Machine learning techniques such as classification have proven to be vital tools in both music information retrieval and music performance, where they are useful for leveraging data to learn and model relationships between low-level features and high-level musical concepts. Explicitly supporting feature extraction and classification in a computer music programming language could lower barriers to musicians applying classification in more performance contexts and encourage exploration of music performance as a unique machine learning application domain. Therefore, we have constructed a framework to execute feature extraction and classification in ChucK, forming a foundation for porting MIR solutions into a real-time performance context as well as developing new solutions directly in the language. We describe this work in depth, prefaced by introductions to music information retrieval, machine learning, and the ChucK language. We present three case studies of applying learning in real-time to performance tasks in ChucK, and we propose that fusing learning abilities with ChucK's real-time, on-the-fly aesthetic suggests exciting new ways of using and interacting with learning algorithms in live computer music performance.

1. INTRODUCTION

Music information retrieval (MIR) involves much active research on making sense of audio, symbolic, and other representations of musical information, so that the vast quantities of musical data available to us may be searched through, visualized, explored, and otherwise better understood and efficiently utilized. Machine learning is widely used in MIR to translate from the low-level features that are easily extracted from musical data to higher-level representations that are more directly relevant to retrieval tasks of interest. While most MIR research focuses on off-line analysis and retrieval (e.g., of audio files stored on disk), numerous MIR problems—including beat tracking, transcription, score following, and identification of instrumentation and gesture—are directly applicable to live music performance as well. Adding explicit support of machine learning to a computer music language thus lowers barriers to implementing MIR solutions in a real-time performance context, making these solutions accessible to a wider user base of composers and performers and encouraging MIR researchers to consider creative applications of their work.

Of course, computer music composers and developers have long employed machine learning techniques to a variety of imaginative ends, only some of which involve the straightforward learning of musical concepts from low-level features. A computer music language that supports employment of "out-of-the-box" machine learning algorithms for traditional learning applications yet also allows easy modification of the algorithms and architecture code *within the language* could facilitate many kinds of creative experimentation.

Finally, we believe the creative and practical possibilities presented by real-time, *on-the-fly* application of machine learning in music performance are under-explored, along with their ramifications for research in machine learning and human-computer interaction. By constructing a foundation for machine learning in ChucK, a language that embraces a strongly-timed, live-coding/on-the-fly aesthetic, we hope to encourage exploration of such issues in research and performance contexts.

In summary, we intend that ChucK offer a high-level programming environment wherein users may easily port existing MIR solutions to a performance context, develop new MIR solutions, and generally apply machine learning algorithms to music performance in novel ways. By facilitating development of systems in ChucK's rapid prototyping and on-the-fly programming paradigm, this work creates new possibilities for exploration in a variety of research, musical, and pedagogical contexts.

To these ends, we begin by acquainting the reader with the general landscape of MIR and the use of machine learning in that field, and we present examples of some ways in which learning has been used in music performance. Building upon the analysis/synthesis paradigm and tools recently integrated into ChucK, we introduce our foundational architecture for incorporating machine learning and MIR tools into this real-time performance language. We present several case studies in which we have used ChucK to build learning algorithms and apply them to musical problems in a real-time, on-the-fly context via a simple and general-purpose user interface.

2. BACKGROUND

2.1. MIR and Machine Learning

MIR is broadly concerned with the analysis, search, discovery, recommendation, and visualization of musical data in its many forms. A few avenues of work

in MIR include transcribing from audio to some type of score representation [10]; identifying tempo and meter [14]; automatically annotating a work with genre [26], mood [16], or social tags[1] [9]; and identifying instruments [12] or performance gestures [24].

As in other areas of multimedia information retrieval, much research in MIR confronts a "semantic gap" between readily computable low-level features and the high-level semantic concepts described above. For example, it is not readily apparent how to translate from common audio features such as the magnitude spectrum, zero-crossing rate, or MFCCs (see [2] and [26] for a good overview of common audio features) to target concepts of interest such as harmony, meter, instrumentation, or genre. Such a semantic gap is often effectively addressed with machine learning methods such as classification, wherein standard algorithms such as k-nearest neighbor [21] (pp. 733–5), neural networks [21] (pp. 736–48), or AdaBoost [22] are used to predict the target concept, or *class*, for vectors of features extracted from the audio (or other data representation). Specifically, in the training stage of classification, the algorithm is presented with a set of training examples consisting of example feature vectors and their associated class labels (i.e., the ground truth). The learning algorithm outputs a prediction rule that is then able to predict the class labels for new data (i.e., feature vectors for which the class label is not known a priori). This is illustrated in Figure 1. For example, a classifier trained for instrument identification might predict the label "flute" or "violin" for a vector of spectral features extracted from a frame of audio.

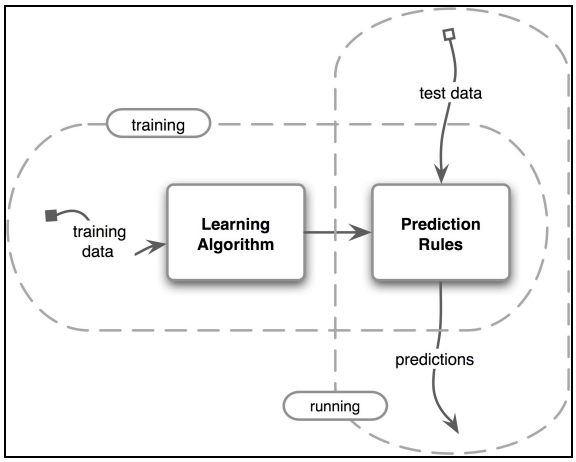

Figure 1. Training and running a classifier.

Learning algorithms differ in the forms of the prediction rules they output, their approaches to constructing those rules from the training data, their flexibility in handling different types of data (e.g., the number of class labels accommodated), and the computational requirements for training and classification. All classifiers, however, share the property of being a data-driven approach to learning to apply labels to feature vectors. The quality of the predictions is dependent in large part on the quantity and quality of the training data. Designing a new system—no matter the form of the raw data or target concept—thus entails choosing a method to extract relevant features, constructing a labeled training data set, choosing a suitable classifier for the problem of interest, and training the classifier on the data. The generality of classification algorithms and their straightforward requirements for application make them an attractive alternative to designing predictors from scratch; in fact, classifiers often provide working solutions to problems that are otherwise intractable (e.g., prediction of fine-grained genre from frame-level audio features).

The first applications of classification in MIR date to the late 1990s [23], and classification has been used in copious MIR systems since then. In fact, all of the example systems referenced in the first paragraph of this section have employed classification as a central component. The MIREX competition [8] offers benchmark datasets against which participating teams may submit classifiers for problems such as genre and mood; now entering its fourth year, it is a testament to MIR's established reliance on classification.

Most MIR research on music itself (as opposed to related social/cultural data or metadata) is restricted to pre-recorded audio, symbolic data files, or other static, non-performative musical representations and contexts. However, there are notable exceptions: following Dannenberg [7], Kapur [13] and Raphael [18] have built impressive systems that carry out MIR tasks in the context of robot/computer accompaniment of a live human performer, and there exist other systems such as [5] that accomplish real-time, non-performance analysis tasks. Furthermore, many of the analysis tasks commonly undertaken in MIR are still relevant to real-time music performance, and in some cases the algorithms can in principle be applied to live, streaming musical data without modification. Clearly, there exists great potential for application and adaptation of MIR techniques by composers and performers.

2.2. Computer Music and Machine Learning

There is a rich history of applications of machine learning techniques in computer music. Neural networks' parallels to human musical perception and cognition make them a compelling tool (see e.g. [25]), and they were used in some of the earliest applications of learning in music, including real-time pitch tracking [19] and gesture identification [31]. Whereas MIR typically uses machine learning as an analysis tool, computer musicians have also used learning algorithms for music creation, for example as tools for generating compositions in a given style [6], expressive performance rendering [4], and interactive improvisation [17][30]. In general, many applications of machine learning in music can be seen as supporting facets of machine musicianship, or implementing

[1] Users of social tagging services such as Flikr, Del.icio.us, and last.fm apply free-form textual "social tags" to content such as pictures, URLs, and music.

components of human musicianship in a computer in order to expand opportunities for composition, interactive performance, and pedagogy (see [20], which also provides numerous examples of approaches to incorporating learning in computer music). Learning thus often serves a similar purpose as in MIR, to bridge the gap between available features and the high-level musical concepts essential for musicianship.

The sort of analysis tasks performed in many computer music learning systems—even those whose end goal is music creation—have much in common with MIR systems. In fact, many of these systems, as well as the score following systems in [13][18] mentioned above, so blur the line between MIR and computer music performance that labeling them in one category or the other is purely a matter of context. However, despite the overlap in goals between the two fields, and the huge potential for application of existing MIR solutions to computer music performance, there is not much sharing of tools between MIR research and computer music. Most MIR systems are designed within a framework that does not easily accommodate music generation. The MARSYAS framework [3] and CLAM library [7] are two exceptional MIR tools that do support sound synthesis, but they are toolkits that are used within C++, not a language designed to fully support computer music. On the other hand, computer music languages' support for the analysis tasks necessary for learning are problematic, in that analysis code must be written outside the language. This impedes tight integration among analysis, synthesis, and control interface code, limits flexibility (the external must be edited and recompiled after modification), and requires the user to have high fluency in another programming language.

3. CHUCK

ChucK is a high-level audio programming language whose programming model promotes a strong awareness of, and provides low-level access and control over, time and concurrency [28]. The operational and programmatic aesthetics of the language encourage rapid experimentation via on-the-fly programming, and its syntax promotes clarity and readability. ChucK also supports a precise concurrent programming model in which processes (called shreds) can be naturally synchronized with both time and data. These aspects of the language lend themselves well (and in different ways) to synthesis, analysis, and the integration of the two.

In the recently developed ChucK analysis framework [29], we introduced the notion of a "Unit Analyzer" (UAna) and a new syntax and semantics specifically tailored to audio analysis programming. These leveraged existing synthesis infrastructure where appropriate, introduced new programming constructs to specify and control audio analysis, and provided basic UAnae for domain transformation (e.g., FFT, DCT), data routing, and simple feature extraction (e.g., Centroid, Flux). The UAna model offers complete and dynamic control over low-level parameters such as FFT windowing and hop size, as well as the ability to craft sophisticated analysis networks. Furthermore, analysis tasks can run in tandem with each other and work cooperatively with synthesis processes in real-time. This programming model and system provided the first steps necessary for on-the-fly learning: the means to perform highly customized feature extraction solely from within the language.

4. INFRASTRUCTURE FOR LEARNING

4.1. Feature Extraction

Desiring to supply standard features used in MIR analysis, we have implemented several new feature extraction UAnae in ChucK. A selection of implemented feature extractors is shown in Table 1. Note that additional composite features can be defined directly in ChucK using these basic features; for example, MFCCs can be computed directly by the MFCC UAna, or constructed by combining more basic UAnae (e.g., FFT, code for Mel frequency scaling, DCT, etc.). This presents the programmer with ready-to-use components as well as the option to make low-level modifications.

FFT	Cross correlation
DCT	Autocorrelation
Spectral Centroid	LPC coefficients
Spectral Flux	MFCC coefficients
Spectral Rolloff	

Table 1. ChucK audio feature extractors.

We have also implemented a **FeatureCollector** UAna class to extract vectors of arbitrary features at synchronous time points, offering easy ability to extract one feature vector per frame or per detected note onset, for example. The code in Figure 2 sets up a UAna patch to extract spectral centroid, spectral flux, and RMS from the mic input for every sliding-window analysis frame.

4.2. Architecture for Classification

We have implemented a basic object-oriented architecture to support classification. Our architecture is based on that of Weka [33], a machine learning and data mining toolkit that is popular in MIR, though we do not use the Weka libraries themselves. A data point, example, or instance (depending on your preferred terminology) is represented by the **Instance** object; an **Instance** contains a feature vector with an associated class label. The **Instances** class represents a full dataset (of zero or more **Instance** objects). All classifiers inherit from the **Classifier** class, which has methods for training (using a training set of **Instances**) and classifying (outputting the predicted class label for an arbitrary **Instance**). Currently implemented classifiers include k-nearest neighbor, AdaBoost.M1 (a multi-class version of Adaboost [22]), and decision stumps (decision trees with a height of one, for use with AdaBoost.M1). **Instance**, **Instances**, and all **Classifier**s are implemented directly in ChucK, so extending this

```
01# // analysis patch
02# FeatureCollector fc => blackhole;
03# adc => FFT fft => Centroid c => fc;
04# fft => Flux f => fc;
05# fft => RMS rms => fc;
06#
07# // (set FFT parameters: window/fft/hop size)
08# // ...
09#
10# // instantiate learner/classifier
11# AdaBoostM1 adaboost;
12#
13# // (train classifier on some data;
14# //  maybe also from mic)
15# // ...
16#
17# // perform feature extraction + classification
18# // in real time on each frame
19# while( true )
20# {
21#     // force computation of all features
22#     fc.upchuck();
23#     // create a new instance; set feature vector
24#     new Instance @=> Instance i;
25#     i.setFeatures( fc.featureVector() );
26#     // classify; print out prediction
27#     <<< adaboost.test( i ) >>>;
28#     // slide analysis window (and time)
29#     HOP_SIZE::samp => now;
30# }
```

Figure 2. Classification in ChucK: performing feature extraction with a **FeatureCollector**, creating a new **Instance** with the extracted features, and using **AdaBoostM1** (trained elsewhere) to classify each frame.

framework requires merely adding the appropriate ChucK code. For example, modifying a classifier implementation or adding a new classifier does not require any coding outside ChucK, recompiling, or even restarting of ChucK code already running. The while-loop in Figure 2 shows a new **Instance** being created from a feature vector and classified by an AdaBoost **Classifier**.

4.3. User Interface for On-the-fly Classification

To demonstrate the usefulness of our system and provide new users with a convenient starting place, we have implemented a simple keyboard-driven interface in ChucK for extracting features, training a binary (two-class) classifier, and performing classification of new inputs on-the-fly. The controls for this interface are described in Table 2, along with the parameters one might wish to set for each step of classification.

The use of a **FeatureCollector** ensures that, in order to change the types of features used for classification, one need only change the definition of the global UGen/UAna patch where the inputs to the **FeatureCollector** are defined. The adherence of all **Classifier**s to a common interface ensures that, in order to change the classifier used, one need only change the instantiation of the classifier and set any classifier-specific parameters (e.g., the number of training rounds used by AdaBoost). It is easy to use one hop size for extracting training data (to control the size and variety of the training set) and another when making predictions on new data (to control the granularity of the predictions, as well as the computational costs of feature extraction in both cases). Furthermore, note that the extracted audio features do not necessarily have to correspond to the features supplied to the classifier; for example, one might wish to train and run the classifier on higher-level statistics such as the means and (co-) variances of features over several subsequent frames, as done in [2], [26]. This involves a trivial code modification of a few lines.

User Action	**Result**	**Parameters (in code)**
Hold '1'	While key is pressed, extracts one feature vector per frame, creates from it a new **Instance** of Class 1, and adds it to the training set.	Hop size
Hold '2'	Same as above, but labels as Class 2.	Hop size
Press '3'	Trains classifier on all training instances recorded so far.	Classifier parameters
Hold '4'	While key is pressed, extracts one feature vector per frame, creates from it a new unlabeled **Instance**, and classifies it with the trained classifier.	Hop size, action taken based on predicted class
Press '5'	Resets classifier and throws out all recorded training examples.	

Table 2. Controls for on-the-fly classification interface.

5. CASE STUDIES

To demonstrate the usefulness of our ChucK classification architecture and motivate discussion of the aesthetic and technical considerations involved in on-the-fly learning, we have implemented three case studies applying classification to simple real-time musical tasks. The complexity of the code in each case is such that an experienced ChucK coder should be able to implement any within a few minutes, starting from the classification framework and user interface described above. The first two tasks were chosen as examples of simple classification problems that might be useful in an improvisatory, interactive live-coding context, and for which training data would be easy to obtain in such a context. The third task is intended as a proxy for an arbitrary audio classification task of greater complexity, where the chosen features are a poor match for the task (as is entirely likely for arbitrary complex tasks chosen on-the-fly), but the learning algorithm is powerful. In each test, we provide rough assessment of qualitative and quantitative metrics of success pertinent to an on-the-fly, interactive context, in lieu of rigorous quantitative performance testing that is unlikely to be relevant to users applying our infrastructure to different problems using different features and different classifiers. In circumstances in which a more traditional

evaluation paradigm is desired, one can use ChucK's file I/O capabilities to load training and testing datasets from saved files and produce accuracy scores given the ground truth. Upcoming work includes augmenting I/O functionality for saving and reloading trained classifiers.

5.1. kNN for Vowel/Consonant and Sung Range Identification

K-nearest neighbor (kNN) classifiers employ a conceptually simple learning method. During the training phase, they merely record the feature and class values of each training instance. To make a prediction for a new instance, kNN finds its *k* nearest neighbors using a Euclidean distance metric in the feature space, then assigns the instance a class label based on a majority vote of those *k* neighbors.

In our first example task, we extracted the RMS, spectral centroid, spectral rolloff, and spectral flux for two classes of speech input: vowels and consonants. Training features were extracted in 1024-sample frames using a 512-sample Hann window and a hop size of 0.1 second. Training data is simple to supply: a speaker can simply say several vowels while holding '1,' then say several consonants while holding '2.' Because no training procedure is required beyond recording the labelled feature vectors, the classifier is immediately ready to use, and it will very accurately classify vowels and consonants in speech input while the user holds '4.'

Instead of merely recording the predicted class label for each new frame, as one might do in a traditional classification system, we broadcast the incoming audio panned to the left or right output channel according to whether it is classified as a vowel or consonant. Classification accuracy is near-perfect if the training data lasts a second or two and covers a good range of inputs (e.g., "aaeeeiiioouuhh", "kkffsssttshhh/*silence*/"), and the panning is acceptably responsive for hop sizes of 0.05 second.

It is trivial to change the classification task performed. For example, we might decide to pan based on the range of the sung pitch (high/low) instead of vowel/consonant. To accomplish this, we can simply replace the timbral features with the magnitude spectrum in the UAna feature extraction patch and retrain the classifier (holding '1' while singing a few low pitches and '2' while singing a few high pitches). For approximately 1 second of training data (i.e., 5 training instances from each class), kNN performs nearly flawlessly on this task.

5.2. kNN for Trackpad Gesture Identification

The user interface code can be used with slight modification to classify data other than audio features. For example, it may be useful to classify gestures from sensor data or other computer inputs. This requires merely defining and extracting features from that data source over time and constructing from them the labelled instances to pass to the classifier.

In our second example, we performed classification of trackpad finger gestures (e.g., line, circle, figure-eight, etc.). We modified the user interface code to extract features for just one training instance every time '1' or '2' was held down, and to accept one new instance to classify every time '4' was held down, where in both cases the trackpad gesture was assumed to take place over the duration of the key press. Our feature vector consisted of ten (*x*,*y*) trackpad position coordinates, sampled uniformly over the duration of the gesture, and normalized to a starting position of (0,0). Using a kNN classifier trained on three instances of each class, we were able to distinguish between horizontal and diagonal lines, between circles and spirals, or between circles and figure-eights nearly perfectly.

5.3. AdaBoost for Artist Identification

Audio artist identification, wherein the performance artist must be classified from the raw audio of a song, is a task more representative of the level of challenge of MIR research problems (artist identification has its own MIREX track, for example). Unlike the examples above, there is no way to guess a means of differentiating between arbitrary artists on the basis of frame-level audio features; some sort of learning or modelling is truly essential.

AdaBoost is a powerful algorithm that been used successfully for artist classification (e.g., by Bergstra et al. in [2], placing 2nd in MIREX 2005). AdaBoost is a "meta-learning" algorithm that repeatedly applies a base classifier to variants of the original learning problem, each with a different weight distribution over the training instances. The output of each of these training rounds is a simple prediction rule, each of which is weighted and combined into a single complex prediction rule output by AdaBoost at the completion of training. In essence, boosting algorithms such as AdaBoost are "based on the observation that finding many rough rules of thumb can be a lot easier than finding a single, highly accurate prediction rule" [22]. The method by which AdaBoost weights training instances and weak predictors allows for theoretical bounds on its training error and classification error under appropriate conditions, and the algorithm also often performs very well in practice on a variety of problems.

Bergstra et al.'s industrial-strength application of AdaBoost to artist classification used decision trees and decision stumps as base learners, in conjunction with a suite of state-of-the-art features. Here, we also boosted on a decision stump, but we merely used the same four timbral features as in Section 5.1. Furthermore, while Bergstra aggregated audio feature statistics over several consecutive frames, we continued to classify a single frame at a time. We also limited AdaBoost to 50 training rounds (a relatively low number).

To provide training data, we played a two-second clip from each of three songs by Led Zeppelin (class 1) and three songs by Joni Mitchell (class 2) (played into the laptop's internal microphone). During training, we extracted the features using the same parameters as in section 5.1, but with a hop size of .05 seconds, therefore

supplying 120 training instances for each class. After 50 training rounds (taking about 5 seconds total), AdaBoost achieved 82.5% accuracy. To test, we played 10 seconds from each of 3 different songs by each artist. AdaBoost achieved 72.3% accuracy in frame-level classification (in fact, accuracy for most songs was much higher, except frames from "Stairway to Heaven" were overwhelmingly classified as Joni Mitchell).

This accuracy is significantly less than Bergstra (77% on a 77-artist dataset), yet impressively better than random when considering the unsuitability of the features, the short training time, and the paltry amount of training data. Depending on the context, accuracy might easily be improved by supplying more data, using more informative features, using feature statistics averaged over multiple frames, allowing AdaBoost to train for more rounds, etc. These modifications could be made on-the-fly according to the desired accuracy and the time available. In the extreme case, one could duplicate Bergstra's implementation in ChucK, train the classifier on a large amount of data, and expect excellent classification accuracy.

6. DISCUSSION OF ON-THE-FLY CLASSIFICATION IN MUSIC

Machine learning research focuses to a great extent on the theoretical properties of classifiers and their implications for accuracy on different categories of problems. Applied machine learning—including work in MIR—is also concerned with discovering how accurately and quickly algorithms perform on real data. Much intuition and knowledge from both areas is directly applicable to on-the-fly classification in music performance. For example, kNN is known to be easily overwhelmed by noisy, irrelevant, or too numerous features [32], making its application problematic if it is not clear which available features will be most relevant to the target concept. AdaBoost may be a wiser choice for such problems, as its accuracy is not hurt by these properties (assuming a suitable base learner).

While standard quantitative metrics such as time and accuracy may be very relevant to applying classification in a music performance context, the tradeoffs among these metrics may be quite different than in more traditional applications. Whereas a MIR researcher may look for the most accurate algorithm whose training time is feasible on the available data (where "feasible" may entail days or even weeks!), a user of on-the-fly classification in music may demand that the training time is absolutely minimal (and predictably so), even at significant costs in prediction accuracy. Additionally, applying machine learning in a music performance context brings to light the importance of more *qualitative* aspects of algorithm behavior: Are the sorts of mistakes a classifier makes musically consequential? aesthetically unforgivable? actually *desirable?* Is the time required to train a classifier to the desired accuracy—more precisely, to an appropriate level of *risk* considering the likelihood and consequences of inaccurate predictions—appropriate for a performance? a rehearsal? solitary experimentation? Is the interface for controlling a classifier suitable to the context and the user? Does there exist adequate means of reasoning about and mitigating time requirements and incurred risk, using the control interface and the available feedback?

Different classifiers, control interfaces, and musical contexts imply different responses to these questions. For example, one finds in practice that it is very easy to iteratively improve kNN's accuracy with our simple user interface by adding training examples on-the-fly. If one notices that the kNN vowel/consonant system does not classify /*ar*/ correctly, one can simply record some new training input of this sound with the desired label and expect the classifier performance to improve from then on without further effort. However, if one wishes to add examples to AdaBoost to improve its performance, one has to re-run the training stage of the algorithm (i.e., repeat all 50 training iterations). Even when the performance context allows for this delay, there is something less gratifying than the immediacy of the train/listen/re-train/listen interaction offered by kNN. On the other hand, the classification time of kNN scales less well with the number of training examples, so an over-eager user who supplies very many examples may overload the system and result in classification computation interfering with audio sample generation. While the examples presented above worked without problem in real-time, the computational feasibility of any particular approach to learning is highly dependent on the user, goal, and context.

Unfortunately, the peculiar requirements for on-the-fly-music classification are not characteristic of problems most typically considered in theoretical or applied machine learning, so there does not exist a standard body of algorithms addressing these requirements, nor established rules of thumb for managing these requirements when building and using such systems. In fact, many qualitatively desirable behaviors, such as the tight interaction offered by kNN, may not even be considered explicitly until users build a system, begin to use it, and find that they are satisfied or dissatisfied. The first implication of this observation is that tools supporting on-the-fly classification in music performance should support rapid prototyping and experimentation, so that users may efficiently explore a range of possible implementations until they find one that "feels good." We strive to accomplish this in ChucK, a language whose design goals have included rapid prototyping ability since its birth, but we plan to continue improving upon prototyping ability by making available a growing set of feature extractors and classifiers that users can employ out-of-the-box. We are currently developing a central repository where users can also contribute their own classifiers and interface code.

The second implication of the uniqueness of this problem space is the need to consider in a principled way the matching of tools (algorithms and their control interfaces) to musical tasks. Framed thus, real-time

classification in performance is an HCI problem, and one that researchers relentlessly explore in music under the umbrella of controller design (see e.g. [27]).

It is safe to say that there exist many tools from machine learning that have great potential to better match the task of real-time music performance than the standard classifiers discussed above. Most obviously, on-line learning algorithms continue learning as data becomes available [11], alleviating the need to re-train from scratch and improving the immediacy of interaction between the algorithm and the user. We venture, however, that the uniqueness of real-time, on-the-fly learning in music performance precludes existing algorithms developed for other fields from effectively addressing all the relevant requirements of this problem space. We are therefore also excited to consider adapting existing algorithms for use especially in this context, for example by devising and exposing musically-relevant, semantic-level control parameters, or by adapting learning objective functions to reflect aspects of the musical context.

7. CONCLUSIONS

We have built upon our prior integration of analysis and synthesis in ChucK to provide a foundation to extract features and learn from them using standard classification algorithms. Using ChucK's extractors for commonly-used features, growing collection of standard classifiers, and extensible architecture for adding new learning algorithms, many MIR algorithms can be easily ported to ChucK without modification. ChucK's support for rapid prototyping also makes it possible for researchers to explore implementation of new MIR algorithms in the language. The same tools for feature extraction, classification, and prototyping are also useful to computer music performers and composers who continue a tradition of employing learning to support facets of machine musicianship and algorithmic composition.

While MIR researchers and musicians already have several choices of toolkits and languages for integrating learning into their work, we are excited that ChucK offers both groups the abilities to perform flexible feature extraction, employ robust standard classification algorithms, and apply state-of-the-art MIR solutions in a widely-used and ever-evolving computer music performance language. We hope this work will significantly lower barriers to applying powerful MIR algorithms for harmonic, rhythmic, structural, and other high-level analysis tasks to live musical performance contexts, where they may open up opportunities for new forms of interaction between computers and humans. Moreover, the code for these tasks is written in ChucK, so users can modify and extend these behaviors without the need for externals, re-compilation, or otherwise interrupting the music.

Our work also naturally facilitates the training and application of classifiers *on-the-fly*, a task for which we have constructed a simple user interface and which we have begun to explore in three case studies. Learning new concepts during performance has rarely been considered in MIR or computer music, but it is a compelling tool for allowing computers to bridge the semantic gap between available audio or gestural features and high-level musical concepts of immediate interest, permitting interaction between humans and computers to occur on this higher level.

In general, the integration of classification tools into a music performance language raises interesting questions about how to manage both quantitative and qualitative aspects of classifier behavior in a performance context. Applying a human-computer interaction perspective to on-the-fly learning in music will motivate our next steps in exploring existing and new learning algorithms. In the meantime, we hope that other researchers and performers will begin to explore ChucK's learning abilities for themselves and become inspired to create new kinds of music, apply existing algorithms in new ways, and otherwise build on the foundation we have established.

ACKNOWLEDGEMENTS

We would like to thank LibXtract for providing an open-source and highly useful set of feature extraction primitives and their implementations, and to gratefully acknowledge the reviewers for their insightful comments. This material is based upon work supported under a National Science Foundation Graduate Research Fellowship.

REFERENCES

[1] Amatriain, X., P. Arumi, and D. Garcia, "CLAM: A framework for efficient and rapid development of cross-platform audio applications," *Proceedings of ACM Multimedia*, Santa Barbara, CA, 2006.

[2] Bergstra, J., N. Casagrande, D. Erhan, D. Eck, and B. Kégl, "Aggregate features and AdaBoost for music classification," *Machine Learning*, vol. 65, pp. 473–84, 2006.

[3] Bray, S., and G. Tzanetakis, "Implicit patching for dataflow-based audio analysis and synthesis," *Proceedings of the International Conference on Music Information Retrieval*, London, UK, 2005.

[4] Bresin, R., "Artificial neural networks based models for automatic performance of musical scores," *Journal of New Music Research*, vol. 27, no. 3, 1998, pp. 239–70.

[5] Casagrande, N., D. Eck, and B. Kégl, "Frame-level audio feature extraction using AdaBoost," *Proceedings of the International Conference on Music Information Retrieval*, London, UK, 2005, pp. 345–50.

[6] Cope, D., *Computers and musical style*. Madison, WI: A-R Editions, Inc. 1991.

[7] Dannenberg, R., "An on-line algorithm for real-time accompaniment," *Proceedings of the International Computer Music Conference*, 1985, 193–8.

[8] Downie, J. S., K. West, A. Ehmann, and E. Vincent, "The 2005 Music Information Retrieval Evaluation

eXchange (MIREX 2005): Preliminary overview," *Proceedings of the International Symposium on Music Information Retrieval*, London, UK, 2005, pp. 320–3.

[9] Eck, D., T. Bertin-Mahieux, and P. Lamere, "Autotagging music using supervised machine learning," *Proceedings of the International Conference on Music Information Retrieval*, Vienna, Austria, 2007.

[10] Ellis, D. P. W., and G. E. Poliner, "Classification-based melody transcription," *Machine Learning*, vol. 65, 2006, pp. 439–56.

[11] Freund, Y., and R. E. Schapire, "Game theory, on-line prediction and boosting," *Proceedings of the Ninth Annual Conference on Computational Learning Theory*, 1996, pp. 325–32.

[12] Fujinaga, I., and K. MacMillan, "Realtime recognition of orchestral instruments," *Proceedings of the International Computer Music Conference*, 2000, pp. 141–3.

[13] Kapur, A., and E. Singer, "A retrieval approach for human/robotic musical performance," *Proceedings of the International Conference on Music Information Retrieval*, 2006, pp. 363–4.

[14] Klapuri, A. P., A. J. Eronen, and J. T. Astola, "Analysis of the meter of acoustic musical signals," *IEEE Transactions on Audio, Speech and Language Processing*, vol. 14, no. 1, 2006.

[15] Lew, M. S., N. Sebe, C. Djeraba, and R. Jain, "Content-based multimedia information retrieval: State of the art and challenges," *ACM Transactions on Multimedia Computing, Communications and Applications*, vol. 2, no. 1, Feb., 2006, pp. 1–19.

[16] Liu, D., L. Lu, and H.-J. Zhang, "Automatic mood detection from acoustic music data," *Proceedings of the International Symposium on Music Information Retrieval*, 2003.

[17] Pachet, F., "The Continuator: Musical interaction with style," *Proceedings of the International Computer Music Conference*, 2002, pp. 211–8.

[18] Raphael, C., "A Bayesian network for real-time musical accompaniment," *Proceedings of Neural Information Processing Systems*, Vancouver, Canada, 2001.

[19] Rodet, X., "What would we like to see our music machines capable of doing?" *Computer Music Journal*, vol. 15, no. 4, Winter 1991, pp. 51–4.

[20] Rowe, R., *Machine musicianship*. Cambridge, MA: The MIT Press, 2001.

[21] Russell, S., and P. Norvig, *Artificial intelligence: A modern approach*, 2nd ed. Upper Saddle River, NJ: Pearson, 2003.

[22] Schapire, R., "The boosting approach to machine learning: An overview," *MSRI Workshop on Nonlinear Estimation and Classification*, Berkeley, CA, 2001.

[23] Scheirer, E., and M. Slaney, "Construction and evaluation of a robust multifeature speech/music discriminator," *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, 1997, pp. 1331–4.

[24] Tindale, A., A. Kapur, G. Tzanetakis, and I. Fujinaga, "Retrieval of percussion gestures using timbre classification techniques," *Proceedings of the International Conference on Music Information Retrieval*, 2004, pp. 541–5.

[25] Todd, P., and D. Loy, *Music and connectionism*. Cambridge, MA: The MIT Press, 1991.

[26] Tzanetakis, G., G. Essl, and P. R. Cook, "Automatic musical genre classification of audio signals," *Proceedings of the International Symposium on Music Information Retrieval*, 2001.

[27] Wanderley, M. M., and N. Orio, "Evaluation of input devices for music expression: Borrowing tools from HCI," *Computer Music Journal*, vol. 26, no. 3, 2002, pp. 62–76.

[28] Wang, G., and P. R. Cook, "ChucK: A concurrent, on-the-fly audio programming language," *Proceedings of the International Computer Music Conference*, 2003.

[29] Wang, G., R. Fiebrink, and P. R. Cook, "Combining analysis and synthesis in the ChucK programming language," *Proceedings of the International Computer Music Conference*, Copenhagen, Denmark, 2007.

[30] Weinberg, G., and S. Driscoll, "Toward robotic musicianship," *Computer Music Journal*, vol. 30, no. 4, Winter 2006, pp. 28–45.

[31] Wessel, D., "Instruments that learn, refined controllers, and source model loudspeakers," *Computer Music Journal*, vol. 15, no. 4, Winter 1991, pp. 82–6.

[32] Wettschereck, D., D. W. Aha, and T. Mohri, "A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms," *Artificial Intelligence Review*, vol. 11, 1997, pp. 273–314.

[33] Witten, I. H., and E. Frank, *Data mining: Practical machine learning tools and techniques*, 2nd ed. San Francisco: Morgan Kaufmann, 2005.