A Meta-Instrument for Interactive, On-the-fly Machine Learning

Rebecca Fiebrink
Department of Computer Science
Princeton University
fiebrink@princeton.edu

Dan Trueman
Department of Music
Princeton University
dtrueman@princeton.edu

Perry R. Cook
Departments of Computer Science & Music
Princeton University
prc@cs.princeton.edu

Abstract
Supervised learning methods have long been used to allow musical interface designers to generate new mappings by example. We propose a method for harnessing machine learning algorithms within a radically interactive paradigm, in which the designer may repeatedly generate examples, train a learner, evaluate outcomes, and modify parameters in real-time within a single software environment. We describe our meta-instrument, the Wekinator, which allows a user to engage in on-the-fly learning using arbitrary control modalities and sound synthesis environments. We provide details regarding the system implementation and discuss our experiences using the Wekinator for experimentation and performance.

Keywords: Machine learning, mapping, tools.

1. Introduction
Copyright remains with the author(s)Joe the musician would like to build a new instrument for musical expression. He has an input modality in mind: perhaps he would like to use his favorite game controller, or dance in front of his webcam. He also has a synthesis algorithm or compositional structure that he would like to drive using these inputs.

Joe sits down at his computer and shows it a few examples of input gestures, along with his desired output parameters for synthesis or compositional controls. He trains a machine learning algorithm to map from inputs to outputs, runs the trained model, and begins to expressively perform his new instrument.

Joe gives the computer a few more examples of input gestures and their corresponding output parameters, re-trains the model, and continues to play. He repeats the process several more times, creating an instrument that becomes more and more complex. Or, he saves the trained algorithm so he can play the instrument later. Or, still unsatisfied, he tries out a different learning algorithm entirely, or changes its parameters, or changes the input features he uses for control.

Joe does all of this in a few minutes, without writing any code. And he does it on stage during a live performance, in front of an audience.

1.1 The Wekinator
We have constructed a new meta-instrument called the Wekinator, which allows musicians, composers, and new instrument designers to interactively train and modify many standard machine learning algorithms in real-time. The Wekinator is a general tool that is not specialized for learning a particular concept, using a particular input controller, or using learning outputs in a particular way. Users are free to choose among a suite of built-in feature extractors for audio, video, and gestural inputs, or they can supply their own feature extractors. They can thus train a learning algorithm to respond to inputs ranging from conducting gestures to vocalizations to custom sensor devices. The user may employ the Wekinator as an example-based mapping creation tool, using the output of the learning algorithm to drive sound synthesis in the environment of her choosing, or she may assign the output some other function. While the general-purpose nature of the Wekinator is an asset, it is particularly distinguished from existing tools by its radically interactive, on-the-fly learning paradigm.

2. Background and Motivation
Machine learning (ML) methods have long been used in the creation of new sound and music interfaces. We are principally interested in the application of supervised ML methods that learn a function (which we will generally refer to as a “model”) relating a set of inputs to outputs, using a training dataset consisting of “true” input/output pairs. Furthermore, we primarily focus here on applying ML methods to creating and modifying controller mapping functions [9], where the input consists of an interface state or gestural controller position, and the output consists of one or more parameters driving sound creation. (We therefore ignore the large body of work on applying ML to computer improvisation and composition.) Such generative approaches to mapping creation, and their tradeoffs with explicit mapping approaches have been compared in [9,16].

The early 1990’s saw the first uses of ML, especially neural networks (NNs) for mapping creation: Lee et al. [12] used NNs for applications including learning mappings from commodity and new music controllers to sound synthesis parameters, and Fels and Hinton [6] built a system for controlling speech synthesis using a data glove. That we cannot even begin to acknowledge all the musical interfaces that have employed NNs as a mapping tool since is a testament to their usefulness, as is the existence of the Pd toolkit for NN mappings by Cont et al. [3]. Matrix-based methods [2] offer yet another tool for generating mappings by examples, though the methods of [2] are unable to learn the highly nonlinear functions of NNs.

While NNs can map inputs into a continuous output space, classification is a form of supervised learning in which the model encodes a function mapping the input space to discrete set of output classes. Classification is appropriate for assigning a category to a gesture, for example [11], and a wide variety of classifier algorithms exist, each with unique tradeoffs and assumptions about the learning problem [18].

Merrill and Paradiso [13] studied users operating an interactive, example-based mapping system for their FlexiGesture input controller, and they found that users preferred building personalized mappings to using an expertly configured static mapping. Their mapping creation system did not use ML to learn the mappings from examples, citing the need for "significant amounts of training data" for successful pattern recognition. However, our recent work experimenting with "on-the-fly" learning of music information retrieval problems [8] suggests that ML techniques might indeed be useful despite very little training data if the concept to be learned is quite focused, a claim also made by Fails and Olsen in [5] and discussed further below. When supervised learning can feasibly be applied, NNs and classifiers offer a more general and flexible set of mapping approaches than Merrill's dynamic time warping gesture classification approach.

The established efficacy of example-based learning for interface mapping has inspired us to create a tool that is general-purpose (not specific to any controller, task, or music environment) and that supports many learning algorithms. Moreover, we are interested in applying ML in a manner that is more radically interactive than existing tools (e.g., [2][3]), offering a unified environment for all stages of the learning process and enabling user interaction to guide and modify the mapping creation in real-time, in an on-the-fly, even performative manner.

3. Wekinator Interaction and Learning

3.1 Interaction with the Wekinator

The Wekinator enables users to rapidly and interactively control ML algorithms by choosing inputs and their features, selecting a learning algorithm and its parameters, creating training example feature/parameter pairs, training the learner, and subjectively and objectively evaluating its performance, all in real-time and in a possibly non-linear sequence of actions. This interactive learning paradigm is illustrated in Figure 1.

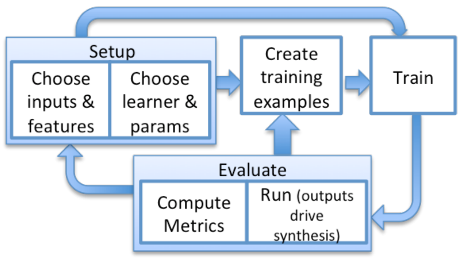


Figure 1: Real-time interactions with the Wekinator

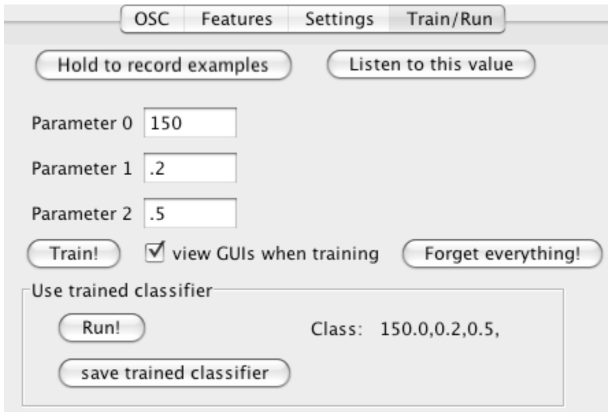
In the setup phase, the user selects a classifier and sets its parameters, as well as specifies which features (also called "attributes") will be extracted from which input sources. These features will fully represent the state of the input controller to the learner; for example, a joystick feature vector might include the current states of all buttons and axes, and an audio feature vector would typically include time- and spectral-domain computations on the audio signal. The choice of features reflects the nature of the mapping the user intends the computer to learn. For example, the spectral centroid audio feature might be extracted for training a learner to differentiate among audio sources with different timbres, whereas FFT bin magnitudes or pitch histograms would be better for differentiation by pitch range or chroma.

In the training example creation phase, the Wekinator extracts the selected features in real-time from the input sources, while the user specifies via a GUI (Figure 2) the desired class labels or function outputs for the current features. For example, a user might enter "100" as the desired output value for a parameter of interest while singing "Ahhh!" into the microphone one or more times. In this phase, the user thus creates and cultivates a dataset that will be used as the training set for the learner.

After creating examples, the user can initiate training, during which the learning algorithm uses the training dataset to build the trained model. The user is able to interact with the training process itself; for example, he can halt a training process that is taking too long, and readjust parameters such as a neural network's learning rate in order to speed up training. The Wekinator is especially designed for scenarios for which this training stage takes at most a number of seconds—an important departure from traditional ML applications, as we discuss later.

To run a trained model, the same features as were extracted to construct the training dataset are again extracted in real-time from the input sources, but now the model computes outputs from the features. These outputs can be used to drive synthesis or compositional parameters of musical code running in real-time. For example, the singing user above may sing "Ahhh!" into the microphone, which will result in a well-trained model outputting "100," which might be used by a synthesis engine to set the frequency of an oscillator to 100Hz.

The user can now evaluate the newly learned model. He might employ traditional objective measures of accuracy,



Parameter 0 150

Parameter 1 .2

Parameter 2 .5

[✓ view GUIs when training]

Class: 150.0,0.2,0.5,

Figure 2: GUI pane for training and running (NN mode, 3 output parameters).

such as cross-validation error. Even more importantly, the user can form judgments of the quality and suitability of the model by running it in real time, observing its response to inputs that may or may not be similar to the training set. He might sing "Ahhh!" at different pitches or volumes to informally test the model's robustness, or he might sing "Oooh!" just to see what happens (serendipitous harmony? horrid noise?). The user can immediately add new examples to the training dataset to correct mistakes or reinforce positive behaviors, and then retrain the learner with the augmented dataset and repeat the evaluation process. Alternatively, the user may decide to use a different learning algorithm or different features in order to achieve a different outcome. Of course, in creative context, the primary goal might not be to train a learner that perfectly implements a user's preferred mapping from inputs to outputs. An instrument designer may simply wish to make the mapping more varied or interesting, or a composer may wish to build increasingly complex mappings from inputs to synthesis parameters in order to aurally explore a synthesis parameter space. A low overhead to interacting with the learning algorithm is beneficial in these circumstances as well.

3.2 Our On-the-fly, Interactive Learning Paradigm

One level of interaction in ML involves providing the user with options to control the learning parameters, for example to edit the architecture of a NN. A second level involves the ability of the computer to extract features and pass them to a running, *pre-trained* model to produce outputs in real-time, enabling musical interaction with a live performer. Obviously, these two definitions of interaction must minimally be satisfied in any ML application for live music, including those in [2,3,6,12].

Agent-based approaches to learning often involve a notion of *learning from the world* in real-time, while also making predictions and taking actions. Musical systems for machine improvisation (e.g., [1]) often learn from a performer in real-time, though musical systems for supervised learning have not taken this approach as far as we know.

The Wekinator offers a more radical degree of interaction than existing supervised learning approaches, which we term "on-the-fly" interactive learning. First, the training set creation happens in real-time and is actively guided by the user. Modifying the training set is an explicit mode of interaction, through which the user can affect the learning outcomes more effectively and predictably than is possible by changing the learning algorithm or its parameters. Second, the entire procedure of modifying the learning process and training set, re-training, and re-evaluating results happens on the order of seconds, not minutes, hours, or longer. Such rapid learning is computationally feasible for many problems of interest, as we discuss in Section 5.2. Moreover, the tight integration of all learning phases of Figure 1 into a single environment and GUI enables the user to interactively experiment with the different learning components and evaluate the effects in an on-the-fly manner. Such an on-the-fly ML system not only supports faster prototyping and exploration than off-line systems, but also opens the door to new performance and composition paradigms, including the live and *performative* training and refinement of a model.

As such, our definition of interactivity most closely matches the ideas of Fails and Olsen [5], who constructed a system for real-time, on-the-fly training of visual object classifiers, in which designers interactively labeled objects on the screen. Fails and Olsen also stress the importance of the speed of the training and interface in supporting true interactivity, and they provide a deeper discussion of how interactive ML's constraints and possibilities differ from traditional ML applications.

4. System Architecture

Figure 3 illustrates the architecture of the Wekinator. Two main components—one in ChucK [17] and one in Java, communicating using OSC [19]—form the core of the system. In order to use the Wekinator for a wide variety of on-the-fly learning tasks, one need use only these components. Users are also able to incorporate their own input devices, feature extractors, and synthesis environments.

4.1.1 Input Setup and Feature Extraction

The Wekinator's built-in feature extractors for hardware controllers and audio are implemented in ChucK. Several of the laptop's native input capabilities, including the trackpad and internal motion sensor, can be used as generic input devices, following our work in [7]. Any USB devices that act as HID joysticks are also usable as input sources; this includes many commodity game controllers as well as custom sensor interfaces built using systems like the CUI[1]. Several common time- and spectral-domain audio features

[1] http://www.create.ucsb.edu/~dano/CUI/

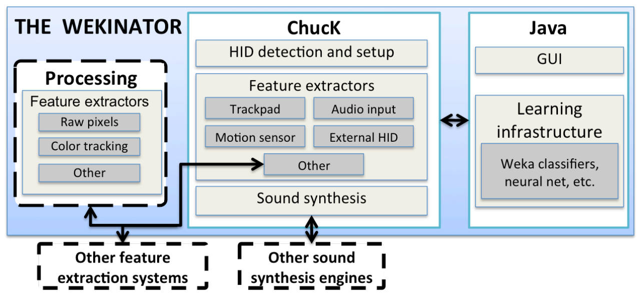


Figure 3: Wekinator architecture. Arrows indicate OSC communication; dotted-line components are optional.

are also available, including FFT bins, spectral centroid, spectral rolloff, and other features common to music information retrieval (MIR) and audio analysis [8].

It is possible to implement additional feature extractors in ChucK, in particular to extract task-specific features from audio or HID devices. For example, we have implemented a set of custom features within ChucK's unit analyzer infrastructure that indicate the relative strength of the spectrum around expected formant frequencies, and used these with the Wekinator to create a personalized vowel recognizer. It is also possible to perform signal conditioning or other transformations on HID input to create features that are less sensitive to jitter, use relative deltas rather than absolute values of a sensor, etc.

One may also use any feature extractor outside of ChucK. For example, we have implemented two simple feature extractors in Processing [14] that use the webcam input: one extracts a simple matrix of edges detected within a 10x10 grid over the camera input, and the other extracts the absolute and relative positions of two differently colored objects held in front of the camera. Any standalone feature extractor must merely communicate its extracted features to ChucK via OSC at the (possibly variable) extraction rate of its choosing.

4.1.2 *Machine Learning*

The Wekinator uses the Weka library [18] for all learning algorithms. Weka is a powerful, free, open-source library written in Java that has enjoyed popularity among music information retrieval researchers [10]. It provides implementations of a wide variety of discrete classifiers and function approximators, including k-nearest neighbor, AdaBoost, decision trees, support vector machines, and NNs. When running a classification algorithm, the Wekinator provides the option of outputting a probability distribution (interpretable as the likelihood of a data point actually belonging to each class) or outputting a single class label (i.e., the class with maximum likelihood).

The Wekinator also supports the learning of multiple concepts simultaneously, for example for the learning of a mapping between controller inputs and *several* synthesis parameters (a many-to-many mapping). In this case, the Wekinator trains one model per parameter. (While it is in theory possible to assign multiple output nodes of a single NN to different synthesis parameters, Weka only supports architectures with a single output node.) The GUI pane in Figure 2 allows the user to specify values for all output parameters simultaneously.

4.1.3 *Synthesis*

The Wekinator comes with several example ChucK synthesis classes that can be driven by the model outputs without modification. A user can write his own ChucK class for using the Wekinator outputs, and it will seamlessly integrate into the existing infrastructure provided it implements our simple API for communicating to the Wekinator its expected number and type of parameters and receiving these parameters from the learner(s). Alternatively, a user can implement sound synthesis in another environment, provided it can perform this parameter communication over OSC.

4.1.4 GUI

Once a user has chosen or implemented the desired synthesis code, and optionally additional feature extraction code, the Wekinator's 4-pane GUI is the single point of interaction with the learning system. The first pane enables the setup and monitoring of OSC communication with the ChucK component of the Wekinator. The second allows the user to specify the features to use and their parameters (e.g., FFT size), create and save configurations for any custom HID devices, and optionally save and reload feature setting configurations. The third pane allows the user to specify the creation of a new model, or to reload a saved, possibly pre-trained model from a file. The fourth (shown in Figure 2 for a NN) allows real-time control over training set creation, training, adjusting model parameters, and running. Its appearance varies depending on the learner; for example, the pane for discrete classifiers includes a button to compute cross-validation accuracy, and the NN pane includes a pop-up window for viewing and editing the network architecture and monitoring the back-propagation learning process and parameters.

5. Discussion

5.1 Mapping as Play

For a practicing composing performer, the process of manually mapping control inputs to audio synthesis and signal processing parameters can be tedious and frustrating. It is generally difficult to predict what kinds of mappings will be "successful," especially when using data-rich audio analysis sources, and the constant mode-shifting between carefully building a complex mapping (which might involve some coding) and then actually auditioning that mapping can be exhausting; in the end, many musicians simply decide to focus their efforts elsewhere.

With instruments like the Bowed-Sensor-Speaker-Array and the R-bow [15], for instance, dozens of sensor parameters and an audio input signal must be mapped (or ignored) to an audio generating algorithm, itself likely having many parameters; exploring this space through manually created mappings is overwhelming and time consuming. What the Wekinator system encourages is a high-level, intuitive approach, where particular mapping "nodes" can be quickly defined via training examples and then the instrument immediately auditioned. Rather than laboriously mode-shifting, the instrument builder now takes part in a playful process of physically interacting with a malleable complex mapping that can be shaped but does not have to be built from the ground up. Furthermore, the surprises that the mapping inevitably generates, while sometimes undesirable, are often inspiring. Being able to save these mappings and revisit them, perhaps modifying them on-the-fly in performance, allows for continuity but also continual evolution.

5.2 On-the-fly Learning in Performance

Six Princeton performers (all of whom were musicians and none of whom were the authors) recently performed a piece, *nets 0*, which employed the Wekinator for on-the-fly learning of controller mappings during the performance.[2] Performers chose their own input devices, resulting in two joysticks, two webcams (one using color tracking and one using the edge detection), one laptop's internal accelerometers, and one hand-made sensor box HID device. Each performer ran the Wekinator on a laptop, connected to a personal 6-channel speaker and subwoofer. Each laptop ran the same ChucK FM synthesis algorithm, which used one Wekinator output to drive the frequency of an oscillator and one output to drive both the gain and reverb wet/dry mix.

The performance began with each player loading up the Wekinator and creating a new NN from scratch. Each player first supplied two initial training examples matching inputs (e.g., joystick positions) to pre-specified values of the two synthesis parameters, trained the network, and began making sound. Over the next five minutes, players interactively built up their controller mappings as they wished, modifying their training set and re-training in an unconstrained manner. Over the second five minutes, players improvised as a group using their established mappings, and the piece culminated with a synchronous, conducted crescendo and decrescendo. As each player created a more complex mapping, the sonic space broadened to include a wider range of sounds and the sonic palette of each player became unique. Also, each player's ability to play expressively grew as time went on.

In this performance, we found the Wekinator to be useable for performers who are not ML experts. We did explain and rehearse the necessary sequence of recording training examples, training, running, etc., until everyone understood well how to play the piece. One performer improvised extensively using her controller with our synthesis algorithm outside the performance, remarking that she enjoyed "exploring a whole new sonic space."

The main source of confusion among performers was the GUI options for changing NN parameters (e.g., learning rate); the piece did not require modifications to these parameters, but performers were confused about their purpose. One can imagine an alternative GUI that exposes "parameters" that are more meaningful to the performers, for example a slider presenting a continuum from "very fast training" to "very accurate training." Such a loss of precise control may be acceptable in exchange for the increased likelihood that performers will feel comfortable trying out the slider and experimenting with its effects. On the other hand, exposing the standard parameters of an algorithm may make the Wekinator an exciting tool to teach students about ML, allowing them to immediately experience the effects of changing parameters.

In rehearsals and performance, on-the-fly learning was always fast enough for real-time interaction, even for the 100 edge detection features. While this represents a dramatic departure from typical ML applications, the ability to learn fast and well enough to provide control is not so surprising. First, real-time creation of the training set results in a few hundred training examples at most, a tiny dataset for ML algorithms designed to accommodate thousands of points. Second, the task is focused in scope: creating a mapping for a single performer to a single synthesis algorithm within a single piece. This is trivial compared to the broader problems for which learning algorithms have been designed, for example decoding handwritten digits written with any handwriting style, or recognizing the face of any human in a photo. Third, the learning is done in a somewhat subjective context, in which it is possible to produce undesirable output (e.g., a painfully high frequency), but where unexpected outcomes are often acceptable and interesting. The explicit goal for each performer throughout our piece was therefore unrelated to traditional metrics of accuracy and generality; performers aimed to create a mapping that was interesting and controllable, and in that everyone was successful. Also, the performers themselves discovered which gestures were learnable and effective. For example, the performer using coarse webcam edge detection learned to play using very large gestures.

5.3 Further Evaluation and Reflection

The Wekinator is not an all-purpose mapping tool, and it is not appropriate for creating especially simple mappings (e.g., one-to-one) or encoding very specific and inflexible mappings that might be better accomplished via explicit coding of the mapping function. It will not work well if the features are a poor match for the task (e.g., using edge detection features to recognize subtle gestures in front of a moving crowd). Weka's learning methods do not incorporate any time domain information, so classification of moving gestures is only possible with custom features

[2] Video at http://wekinator.cs.princeton.edu/nets0/

that take time-domain behavior into account (e.g., sensor deltas) or with another learning system (e.g., hidden Markov models).

On the other hand, the Wekinator is a powerful tool for creating mappings very quickly, for any input features and synthesis method. And its uses go beyond mapping creation: it seems to be a promising tool for the exploration of large parameter spaces for synthesis algorithms such as physical models. Also, using appropriate audio features, it can be used for on-the-fly learning of high-level musical concepts such as pitch, instrumentation, or style. For more complicated learning problems of any nature, one could use Weka's own GUI to train a learner off-line on a larger dataset (such as those used in music information retrieval benchmarking [4]) then load the model into the Wekinator to run on real-time input. Furthermore, the output of the Wekinator could be applied to interactive video or any other systems capable of communicating via OSC.

Future work may further improve the Wekinator for musical contexts by incorporating the ability to constrain which features are used to learn which outputs (e.g., to support one-to-one and one-to-many mappings), and to allow the use of different learning methods for different outputs (e.g., to use a discrete classifier for one output and a neural network for another). The ability to map certain controller inputs to controlling the GUI itself would also offer practical benefits for on-the-fly mapping of controllers that require two hands.

6. Conclusions and Future Work

We have presented the Wekinator, a general-purpose system for radically interactive, on-the-fly machine learning for controller mapping creation and other musical applications. Our experiences with the Wekinator have reinforced our conviction that on-the-fly learning in music is an exciting new paradigm that enables new performance methods and experiences, even with relatively old learning algorithms and synthesis methods. We look forward to continuing to investigate the implications of this new paradigm from the perspectives of human-computer interaction, music performance and composition, and machine learning.

The Wekinator is available to download at http://wekinator.cs.princeton.edu/.

7. Acknowledgments

We thank the autumn 2008 LAP seminar participants for their feedback. This material is based upon work supported under a National Science Foundation Graduate Research Fellowship. This work is also supported by a John D. and Catherine T. MacArthur Foundation Digital Media and Learning grant.

References

[1] G. Assayag, G. Bloch, M. Chemillier, A. Cont, and S. Dubnov, "OMax Brothers: A dynamic tyopology of agents for improvization learning," *ACM Wkshp. Audio and Music Computing Multimedia*, 2006, pp. 125-132.

[2] F. Bevilacqua, R. Müller, and N. Schnell, "MnM: A Max/MSP mapping toolbox," *NIME*, 2005, pp. 85-88.

[3] A. Cont, T. Coduys, and C. Henry, "Real-time gesture mapping in Pd environment using neural networks," *NIME,* 2004, pp. 39-42.

[4] J. S. Downie, K. West, A. Ehmann, and E. Vincent, "The 2005 music information retrieval evaluation exchange (MIREX 2005): Preliminary overview," *Intl. Conf. on Music Information Retrieval (ISMIR)*, 2005, pp. 320-323.

[5] J. Fails and D. R. Olsen, Jr., "Interactive machine learning," *Intl. Conf. on Intelligent User Interfaces*, 2003, pp. 39-45.

[6] S. S. Fels and G. E. Hinton, "Glove-Talk: A neural network interface between a data-glove and a speech synthesizer," *IEEE Trans. on Neural Networks*, vol. 4, 1993.

[7] R. Fiebrink, G. Wang, and P. R. Cook, "Don't forget the laptop: Using native input capabilities for expressive musical control," *NIME,* 2007, pp. 164-167.

[8] R. Fiebrink, G. Wang, and P. R. Cook, "Support for MIR prototyping and real-time applications in the ChucK programming language," *Intl. Conf. on Music Information Retrieval (ISMIR)*, 2008, pp. 153-158.

[9] A. Hunt and M. M. Wanderley, "Mapping performer parameters to synthesis engines," *Organised Sound*, vol. 7, pp. 97-108, 2002.

[10] P. Lamere, "The tools we use," *http://www.music-ir.org/evaluation/tools.html*, 2005.

[11] M. Lee, A. Freed, and D. Wessel, "Neural networks for simultaneous classification and parameter estimation in musical instrument control," *Adaptive and Learning Systems*, vol. 1706, pp. 244-55, 1992.

[12] M. Lee, A. Freed, and D. Wessel, "Real-time neural network processing of gestural and acoustic signals," *ICMC*, 1991, pp. 277-280.

[13] D. J. Merrill and J. A. Paradiso, "Personalization, expressivity, and learnability of an implicit mapping strategy for physical interfaces," *Extended Abstracts: Human Factors in Computing Systems* (CHI'05), 2005, pp. 2152-2161.

[14] C. Reas and B. Fry, "Processing: A learning environment for creating interactive web graphics," *Intl. Conf. on Computer Graphics and Interactive Techniques (SIGGRAPH)*, 2003.

[15] D. Trueman and P. R. Cook, "BoSSA: The deconstructed violin reconstructed," *Journal of New Music Research*, vol. 29, no. 2, 2000, pp. 121-130.

[16] M. M. Wanderley, ed. "Mapping strategies in real-time computer music," *Organised Sound,* vol. 7, 2002.

[17] G. Wang and P. R. Cook, "ChucK: A concurrent, on-the-fly audio programming language," *ICMC,* 2003.

[18] I. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd ed. San Francisco: Morgan Kaufmann, 2005.

[19] M. Wright and A. Freed, "Open sound control: A new protocol for communicating with sound synthesizers," *ICMC,* 1997.