ml.lib: Robust, Cross-platform, Open-source Machine Learning for Max and Pure Data

Jamie Bullock
Birmingham Conservatoire
Birmingham City University
Paradise Place, B3 3HG
jamie.bullock@bcu.ac.uk

Ali Momeni
Carnegie Mellon University
CFA 300 – 5000 Forbes Ave
Pittsburgh, PA 15213
momeni@cmu.edu

ABSTRACT

This paper documents the development of ml.lib: a set of open-source tools designed for employing a wide range of machine learning techniques within two popular real-time programming environments, namely Max and Pure Data. ml.lib is a cross-platform, lightweight wrapper around Nick Gillian's Gesture Recognition Toolkit, a C++ library that includes a wide range of data processing and machine learning techniques. ml.lib adapts these techniques for real-time use within popular dataflow IDEs, allowing instrument designers and performers to integrate robust learning, classification and mapping approaches within their existing workflows. ml.lib has been carefully designed to allow users to experiment with and incorporate machine learning techniques within an interactive arts context with minimal prior knowledge. A simple, logical and consistent, scalable interface has been provided across over sixteen externals in order to maximize learnability and discoverability. A focus on portability and maintainability has enabled ml.lib to support a range of computing architectures—including ARM—and operating systems such as Mac OS, GNU/Linux and Windows, making it the most comprehensive machine learning implementation available for Max and Pure Data.

Author Keywords

Machine Learning, Max, Pure Data, Gesture, Classification, Mapping, Artificial Neural Networks, Support Vector Machines, Regression

ACM Classification

I.2.6 [Artificial Intelligence] Induction, H.5.5 [Information Interfaces and Presentation] Sound and Music Computing.

1. INTRODUCTION

The term 'Machine Learning' refers to a scientific discipline and associated range of techniques that explore the construction and study of algorithms that can 'learn' from data through induction or 'by example' [21]. Typically machine learning techniques are 'black box' systems that can deduce appropriate outputs from given inputs based on a statistical model generated from sufficient and appropriate training data. Supervised machine learning algorithms take a set of labeled feature vectors (lists of values that describe features of a class), which are used to 'train' the machine learning algorithm and generate a model. Once trained, a classification system can output an estimate for the class of unlabeled feature vectors. In the case of regression algorithms, a continuous value is given as output rather than a discrete class. Unsupervised machine learning algorithms take a set of unlabeled feature vectors and partition them into clusters based on some measure of similarity.

Dataflow programming languages offer several important advantages for the NIME community and its neighbors: first, they allow artists, designers and researchers without a deep background in computer science to experiment with complex computational ideas and develop intricate systems for audio and video analysis, synthesis, gestural control and more; second, in comparison with scripting based languages, dataflow programming languages allow for very rapid prototyping of multi-media software, albeit the mastery of these environments requires significant time and investment from the practitioners; third, they allow for integration of a wide range media (e.g. sound, image, physical computing, etc.) within the same environment and with minimal modifications to the same blocks of code [31]; fourth, by hiding certain low-level considerations of control programming (e.g. threading, garbage collection, etc.) dataflow IDEs allow for much more intricate control of time and rather intricate design of time-based behavior; finally, the standard practice for Max and Pure Data (PD) externals—compiled C/C++ code that adds functionality to the standard distributions—to be accompanied by a *help file*—an exemplary program that documents the external and offers and interactive example—make for a very suitable way to introduce new users to machine learning; we have therefore invested significant time in producing *help files* that convey a basic understanding of what each of the ml.lib externals does.

The most widely used machine learning algorithms have long been standardized and bindings in a broad range of programming languages are available. As an example: the most commonly used support vector machine library, Chang et al.'s libsvm [5] was first released in 2001 and iterated several times in the following decade, and cited over 20,000 times according to Google Scholar. Its usage is pervasive throughout human-computer interaction research. While a number of researchers within the NIME community have documented their experimentation with machine learning techniques (see BACKGROUND), development remains scattered; a comprehensive, open-source, cross-platform and user-friendly package for working with machine learning within dataflow environments does not exist. In addition to this lack, the authors' experiences as teachers have also been an important motivation for this project: we note that despite the level of ambition and sophistication in our students' approaches to designing NIME, their approach to the general problem of 'mapping' remains primarily within the realm of arithmetic or discrete calculus at best. The documentation provided with ml.lib therefore reframes some common mapping exercises from the classroom (e.g. recognizing the orientation of a mobile phone based on its three-dimensional accelerometer data) that require many arithmetic operations, as trivial machine learning problems in an effort to convey that machine learning *is* in fact approachable for a broad range of users.

2. BACKGROUND

There is extensive use of machine learning within the domain of sound and music research. Recent contributions to this field include several categories of applications: gesture analysis, mapping and

control of sound synthesis [2, 4, 12, 14, 16, 20, 22, 29] parsing and segmentation [3], and algorithmic composition [6]. A number of existing projects implement various machine learning techniques in Max. As early as 1991, Wessel et al. implemented a real-tine artificial neural network for control of synthesis parameters in Max [30]. Later work from CNMAT includes Schmeder's use of support vector machines applied to pitch predictions [27]. The MnM toolbox from Bevilacqua et al. implements 'multidimensional linear mapping […] as basic module to build complex n-to-m mapping'; the MnM package of Max externals also includes a principal component analysis module which can also be of use in creating complex real-time mappings [1]. It is worth noting that MnM relies on FTM—a shared library for Max for static and dynamic creation of complex data structures and therefore requires an additional step for integration into a user's workflow, a step that may be non-trivial depending on the complexity of the user's existing patch. Cont et al.'s range of work-leading to the development of the sophisticated score-following engine *Antescofo* [7, 8, 9, 11] are among the most advanced examples of machine learning at work within the Max environment. Cont et al. also created neural network implementation for PD, applied to gesture mapping [10].Smith and Garnett developed a machine learning library for Max that implements adaptive resonance theory, self-organizing maps and spatial encoding [28].

A number of externals also exist for the PD environment. The most coherent and widely-used being the ANN library by Davide Morelli[1]. This consists of a multilayer perceptron and a time-delay network, implemented as a wrapper around the widely-used FANN library [24] and a self-organizing map implementation, featuring Instar, Outstar and Kohonen learning rules. A Genetic Algorithm implementation has been developed by Georg Holzman using the flext framework, and is therefore available for both Max and PD. There also exists a k-NN (k's nearest neighbor) external, originally developed by Fujinaga and MacMillan [15] and now maintained by Jamie Bullock. A plan was proposed to develop an SVM external as part of the 2009 Google Summer of Code[2], but to the knowledge of the current authors, this was never realized.

3. IMPLEMENTATION

Our design goals in implementing a suite of machine learning externals are as follows:

- To provide an exhaustive range of machine learning techniques for Max and PD
- To support the main hardware and software platforms supported by Max and PD
- To make machine learning techniques usable and accessible, even for users with no prior knowledge
- To be efficient enough to run classification or regression in firm real-time on CPU's from 700 MHz
- To develop a 'standard' implementation that would be widely adopted and be maintained for the foreseeable future

In addition to these design goals, a number of usage assumptions were also made:

- That externals would operate in a context allowing read / write to disk, allowing save / load of data models and other state
- That users would be responsible for providing appropriate pre- and post-processing steps, e.g. filtering, normalization

3.1. GRT

Given the range of existing machine learning libraries available for C and C++, it was decided that given the limited development resources available, the best approach would be to develop a wrapper around an existing library rather than starting from scratch. An initial survey was conducted and a range of libraries were considered including Dlib[3], mlpack[4] and Shark[5]. We also considered using a collection of C libraries for example libsvm (for Support Vector Machines). After considering the pros and cons of each library, we decided to base ml.lib on the Gesture Recognition Toolkit by Nick Gillian [17] due to its wide range of implemented algorithms, simple design, straightforward C++ interface, pre- and post-processing functions and orientation towards artistic applications, specifically real-time gesture analysis.

3.2. flext

Max and PD both provide C APIs for developing external objects. Whilst the APIs are superficially similar, there are enough differences to mean that in supporting both environments, strategies must be developed for effective code reuse. One approach is to use C macros to conditionally include environment-specific code blocks. This may be sufficient for smaller projects, but for larger projects it degrades readability and creates an unnecessary maintenance burden. An alternative approach is to use the flext API, by Thomas Grill [18], an abstract object-oriented C++ interface that provides a common layer, compatible with both Max and PD. flext is a compile-time dependency meaning that it places no additional installation burden on the end user. flext also has the advantage that through a relatively modern OOP style, and a set of convenience functions, it enables the programmer to write leaner, more readable and more maintainable code than is possible with the conventional C APIs for Max and PD.

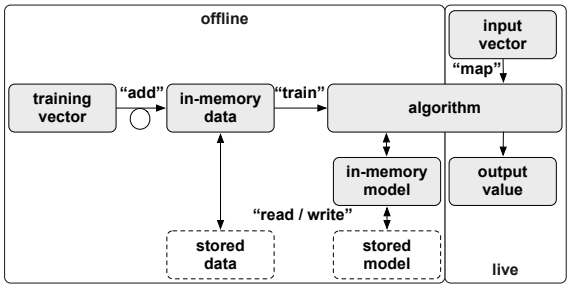

Fig. 1 ml.lib common workflow (object messages in quotes)

3.3. A Maintainable 'DRY' Wrapper

One of the goals of the project has been to develop a library that is maintainable, and can be supported in the long term. The first step in achieving this has been to make the source code available under the GNU General Public License version 2 in a public GitHub repository. This provides a well-managed workflow enabling users and developers to file support issues, and to easily contribute to the project through GitHub 'pull requests', which means patches can be reviewed before being incorpo-

[1] http://bit.ly/morelli_ann

[2] http://bit.ly/pd_svm

[3] http://dlib.net/ml.html

[4] http://www.mlpack.org

[5] http://image.diku.dk/shark

rated into the codebase, whilst the GPL license forbids closed-source forks that may prevent fixes and enhancements being contributed back to their upstream sources.

Another strategy used to ensure maintainability was to adhere strongly to DRY (don't repeat yourself) principles in the development of the wrapper code [19]. This was achieved by developing a number of generic abstract base classes (ml_base, ml, ml_classification and ml_regression) implementing functionality common to the majority of wrapped classes in the GRT library. These classes exploit C++'s runtime polymorphism to call common child class methods in GRT through a reference to a base class instance returned by a concrete child. That is: ml_classification and ml_regression must both implement the pure virtual method get_MLBase_instance() and all children of ml_classification and ml_regression must implement the pure virtual methods get_Classifier_instance() and get_Regressifier_instance() respectively. This means that all common functionality can be implemented in ml_base, by calling methods through a reference to GRT::MLBase from which the majority of GRT classes derive. Only algorithm-specific attributes and methods are implemented in children of ml_classification and ml_regression, making the wrapper code very lean, readable and keeping repetition to a minimum. The current ratio of wrapper code to original sources is 5k SLOC to 41k SLOC or approximately 1:10.

4. LIBRARY DESIGN

From an end-user perspective, we aimed to provide the best possible experience by maximizing learnability and discoverability within the ml.lib library. This was achieved by establishing a convention of consistent and logical object, message and attribute naming, and by designing a simple and consistent workflow across common object groups. In some cases, it was sufficient to follow the well thought-out patterns established in GRT, but in others further abstraction was necessary. Furthermore, the aim was not simply to wrap GRT, exposing every detail of the GRT API, but rather to provide a somewhat abstracted set of objects conforming to the idioms and user expectations of dataflow environments. ml.lib objects follow the naming convention ml.* where '*' is an abbreviated form of the algorithm implemented by the object.

Objects fall into one of six categories:

Pre-processing: pre-process data prior to used as input to a classification or regression object

Post-processing: post-process data after being output from a classification or regression object

Feature extraction: extract 'features' from control data. Feature vectors can be used as input to classification or regression objects

Classification: take feature vectors as input, and output a value representing the class of the input. For example an object detecting hand position might output 0 for left, 1 for right, 2 for top and 3 for bottom.

Regression: perform an *M* x *N* mapping between an input vector and an output vector with one or more dimensions. For example an object may map *x* and *y* dimensions of hand position to a single dimension representing the distance from origin (0, 0)

Clustering: partition *N* unlabeled vectors into *M* clusters

At the time of writing, classification, regression and several feature extraction algorithms are implemented. Full details of these are outlined in section 6.

In order to reduce complexity, and conform to usability best practices a simple modeless workflow was devised (Fig. 1). This workflow was based on an abstraction of the simplest possible steps required to add exemplars to a machine learning system, train it, and use the trained model to map unknown inputs to outputs. The aim of this workflow is to make machine learning techniques usable on *musical* problems and by users who are not machine learning experts. The ability to save and load both data sets and trained models allows for easy comparison and debugging between data sets and algorithms. Although threaded training is not yet implemented, training is very quick—perceptually instantaneous in our tests (section 6)—allowing for rapid 'exploratory' iterations for experimenting with algorithm parameters as described in [13]. All classification and regression objects in the library have exactly one inlet and two outlets. The inlet accepts a number of common 'method' messages:

add <class> <values…> A method used to add training vectors as exemplars for the machine learning algorithm, where <class> is an integer identifying the class corresponding to a vector of 2 or more values

train Once an adequate number of training vectors have been added, this method is used to train the algorithm and generate an in-memory model

write <path> Write the current in-memory training data and / or model (if available) to file

read <path> Read the training data and / or model into memory from file given by <path>

map <values…> Perform classification or regression on the input vector given by <values…> and send the result to the left outlet

clear Remove in-memory training data and / or model

help Post information to the Max or PD console about supported methods and attributes for the current object

For algorithms that deal explicitly with time series, such as Dynamic Time Warping, an additional record message is used to place the object in record mode (the one exception to the modeless design)—vectors add-ed between record 1 and record 0 are treated as contiguous time series. The inlet can also be used for setting the state of attributes. Here, the term 'attributes' refers to 'stored object state' following the flext convention, not Max attributes. flext attributes can additionally be set using object creation arguments, e.g. @scaling 1. There are many object-specific attributes corresponding to the unique configuration parameters of each ML algorithm. The only common attributes are scaling (0/1), which toggles automatic pre-scaling for input vectors and probs (0/1), which toggles the output of class probabilities to the right outlet. The left outlet is used for classification and regression output values.

5. MAX AND PURE DATA EXTERNALS

Given that ml.lib primarily wraps the functionality of GRT, the following sections (5.1–5.14) are based on the excellent GRT official documentation[6] (as indicated by single quotes), used here with kind permission of the original author.

[6] http://www.nickgillian.com/software/grt

5.1 ml.adaboost: Adaptive Boosting

'AdaBoost (Adaptive Boosting) is a powerful classifier that works well on both basic and more complex recognition problems. AdaBoost works by creating a highly accurate classifier by combining many relatively weak and inaccurate classifiers. AdaBoost therefore acts as a meta algorithm, which allows you to use it as a wrapper for other classifiers.'

5.2 ml.dtree: Decision Trees

'Decision Trees are conceptually simple classifiers that work well on even complex classification tasks. Decision Trees partition the feature space into a set of rectangular regions, classifying a new datum by finding which region it belongs to.' 'A decision tree is a flowchart-like structure in which each internal node represents a "test" on an attribute (e.g. whether a coin flip comes up heads or tails), each branch represents the outcome of the test and each leaf node represents a class label (decision taken after computing all attributes). The paths from root to leaf represents classification rules.'

5.3 ml.dtw: dynamic time warping

'The DTW algorithm is a supervised learning algorithm that can be used to classify any type of N-dimensional, temporal signal. The DTW algorithm works by creating a template time series for each gesture that needs to be recognized, and then warping the real-time signals to each of the templates to find the best match. The DTW algorithm also computes rejection thresholds that enable the algorithm to automatically reject sensor values that are not the K gestures the algorithm has been trained to recognize (without being explicitly told during the prediction phase if a gesture is, or is not, being performed). In time series analysis, dynamic time warping (DTW) is an algorithm for measuring similarity between two temporal sequences, which may vary in time or speed. For instance, similarities in walking patterns could be detected using DTW, even if one person was walking faster than the other, or if there were accelerations and decelerations during the course of an observation. DTW has been applied to temporal sequences of video, audio, and graphics data—indeed, any data which can be turned into a linear sequence can be analyzed with DTW.'

5.4 ml.gmm: Gaussian mixture models

'The Gaussian Mixture Model Classifier (GMM) is basic but useful supervised learning classification algorithm that can be used to classify a wide variety of N-dimensional signals.'

5.5 ml.hmm: hidden Markov models

'Hidden Markov Models are powerful classifiers that work well on temporal classification problems when you have a large training dataset.'

5.6 ml.knn: k-nearest neighbors

'The K-Nearest Neighbor (KNN) Classifier is a simple classifier that works well on basic recognition problems, however it can be slow for real-time prediction if there are a large number of training examples and is not robust to noisy data. In pattern recognition, the *k*-Nearest Neighbors algorithm (or *k*-NN for short) is a non-parametric method used for classification and regression. In both cases, the input consists of the *k* closest training examples in the feature space.'

5.7 ml.linreg: linear regression

'In statistics, linear regression is an approach for modeling the relationship between a scalar dependent variable y and one or more explanatory variables denoted *X*. The case of one explanatory variable is called simple linear regression. For more than one explanatory variable, the process is called multiple linear regression.'

5.8 ml.logreg: logistic regression

'Logistic regression measures the relationship between the categorical dependent variable and one or more independent variables, which are usually (but not necessarily) continuous, by using probability scores as the predicted values of the dependent variable.'

5.9 ml.mindist: minimum distance

'The MinDist algorithm is a supervised learning algorithm that can be used to classify any type of *N*-dimensional signal. The MinDist algorithm works by fitting *M* clusters to the data from each class during the training phase. A new sample is then classified by finding the class that has the cluster with the minimum distance (Euclidean) to the new sample.'

5.10 ml.mlp: multi-layer perceptron

'The MLP algorithm is a supervised learning algorithm that can be used for both classification and regression for any type of *N*-dimensional signal. A multilayer perceptron (MLP) is a feedforward artificial neural network model that maps sets of input data onto a set of appropriate outputs. A MLP consists of multiple layers of nodes in a directed graph, with each layer fully connected to the next one. '

5.11 ml.randforest: random forests

'Random Forests are an ensemble learning method that operate by building a number of decision trees at training time and outputting the class with the majority vote over all the trees in the ensemble. Random forests are an ensemble learning method for classification (and regression) that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes output by individual trees. The algorithm for inducing a random forest was developed by Leo Breiman and Adele Cutler, and 'Random Forests' is their trademark. The term came from random decision forests that was first proposed by Tin Kam Ho of Bell Labs in 1995.'

5.12 ml.svm: support vector machines

'In machine learning, support vector machines (SVMs, also support vector networks) are supervised learning models with associated learning algorithms that analyze data and recognize patterns, used for classification and regression analysis. Given a set of training examples, each marked as belonging to one of two categories, an SVM training algorithm builds a model that assigns new examples into one category or the other, making it a non-probabilistic binary linear classifier. An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall on.'

6. INITIAL TESTING

A series of initial experiments were performed to test the functionality of ml.lib within Max and PD for creative applications. These experiments were designed to be repeatable by students, artists, designers and other potential users with minimal requirements besides Max and PD. The first three examples utilize sensor data—specifically the three-axis accelerometer data—from a mobile phone. The tests were conducted by Momeni in the ArtFab, a mixed media design and fabrication lab at Car-

negie Mellon University. We employed the app TouchOSC[7], which sends sensor values over UDP at a rate of 50Hz. The first three test applications classify the orientation of the phone in space, classify a continuous time-based gesture performed by the user with the phone in his/her hand, and allow continuous control of a complex parameterized audio-synthesis. The final example implements a powerful form of swept frequency acoustic sensing that allows the user to transform a passive rigid object into a gestural controller without any wires or typical sensors. These test applications were implemented in Max and PD, using only native objects and ml.lib, thus allowing our students to employ these techniques on a range of platforms including Max on personal computers and PD on Raspberry Pi[8].

6.1 Orientation Classification

We developed a test application that allows the user to train a support vector machine in order to classify several orientations of a mobile phone based on its sensor data. In this example, the feature vectors put into the SVM are three-dimensional vectors—or three-element lists in Max and PD—made up of the x/y/z accelerometer data from the phone and the classification indicates one of several orientations; this is therefore an example of a 3-to-1 dimension mapping. The user trains the SVM by providing a number of examples for each orientation (about 10 examples is sufficient for very accurate classification). While this classification task is perfectly feasible with traditional approaches using arithmetic scaling, our approach requires no pre- or post-processing of the data, thereby rendering the classification task rather trivial for the user. This application is provided with the help-patch for the external ml.svm as a subpatch named 'test'; the help patch also gives reference to a demonstration video shared on YouTube[9].

6.2 Gesture Classification

We developed a test application that allows the user to train a dynamic time warping algorithm in order to classify several gestures performed by the user with a mobile phone in his/her hand. In this example, the input to the DTW engine is three-dimensional time series—or a set of three-element lists in Max and PD—made up of the x/y/z accelerometer data from the phone, sampled at equal intervals. After the system is trained by providing an example of each gesture, the applications can accurately classify new gestures. A noteworthy benefit of employing DTW for this task is the system's invariance in relation to the speed with which the gesture is performed; slower or faster performances of the same gesture are recognized as well. This application is provided with the help-patch for the external ml.dtw as a subpatch named 'test'; the help patch also gives reference to a demonstration video shared on YouTube[9].

6.3 Control of synthesis parameters

We developed a test application that allows the user to train an artificial neural network (a multilayer perceptron or MLP) to generate parameters that control phase-aligned-formant synthesis [25]. The synthesis is implemented as an abstraction in Max and PD with seven inputs (fundamental frequency, amplitude, filter center frequency, filter bandwidth, vibrato depth, vibrato frequency, and frequency shift). In this example, the input to the MLP is three-dimensional vector made up of the x/y/z accelerometer data from the mobile. The output of the network is a seven-dimensional vector corresponding to the synthesis parameters. A training example for this application consists of the three-dimensional feature vector and the corresponding seven-dimensional desired output vector (i.e the synthesis paramters). The system is therefore performing a 3-to-7 dimensional mapping. This approach to n-to-m mapping provides a useful counterpart to a weighted-interpolation technique used to similar ends [23] as it provides opportunities for extrapolation, i.e. generating synthesis parameters that are outside of the range of possibilities achieved by mixing the predefined examples arithmetically. In our experience, these extrapolations can be very useful compositional discovery tools for synthesis schemes that have many parameters whose influence on the resulting sound is highly inter-related. This application is provided with the help-patch for the external ml.mlp as a subpatch named 'test'; the help patch also gives reference to a demonstration video shared on YouTube[9].

6.4 Acoustic Swept Frequency Sensing

We developed a test application that implements Ono et al.'s Touch and Activate [25] in Max and PD. This technique allows users to transform passive rigid objects (e.g. a ceramic bowl, a wooden table, a series of Lego blocks) into sensing objects using swept-frequency acoustic sensing. In short, the technique involves injecting a swept-frequency signal above human hearing into the object using a piezo element, and re-capturing the signal using a second piezo element. As Ono et al. show, touching the object in different ways affects the spectral content of the incoming signal in ways that an SVM can classify very accurately. This application is provided as a stand-alone patch and distributed with the ml.lib package; the patch also gives reference to a demonstration video shared on YouTube[9].

7. CONCLUSIONS AND FUTURE WORK

In this paper we have described the design, development and testing of a new library of machine learning externals for the Max and PD environments based on the C++ Gesture Recognition Toolkit. Our library, ml.lib provides a robust and efficient way to incorporate machine learning techniques into standard dataflow languages, and provides a sufficiently simple and consistent interface to make it accessible to diverse range of users, including those with little machine learning experience, making it an ideal tool for teaching environments. Our initial tests within the context of creative practice show that ml.lib is capable of handling a number of common use cases.

At the time of writing ml.lib includes over 16 externals, which wrap most of the classification and regression classes provided by GRT, as well as including several custom externals for peak detection and minima and maxima extraction. Future work will include the development of wrappers for a wider range of functionality provided by GRT, which includes clustering, pre- and post-processing and feature extraction. Also, whilst ml.lib currently provides extensive documentation in the form of help files and online information, we plan to supplement this with a range of use-case examples and tutorials. It is our aim to build a vibrant user community around the library, and to provide a forum for user-contributed content relating to the library.

Finally, we plan to assess the user requirement for on-the-fly algorithm training. This would allow users to 'add' exemplars to ml.lib objects during live performance and to 'train' the algorithms concurrently with other processes running in Max and PD, even for large datasets. This would be achieved by providing an asynchronous 'train' method implemented using a separate thread. Additional future work will include more extensive end-user testing, the provision of in-environment unit tests, and

[7] http://hexler.net/software/touchosc

[8] http://www.raspberrypi.org

[9] http://bit.ly/artfab_video

'sanity checks' comparing outputs from ml.lib objects to results from the underlying GRT implementation.

8. ACKNOWLEDGMENTS

Many thanks to Nick Gillian and Thomas Grill for their support in the development of ml.lib.

9. REFERENCES

[1] Bevilacqua, F., Müller, R., & Schnell, N. (2005). MnM: a Max/MSP mapping toolbox. *Proceedings of the conference on New Interfaces for Musical Expression*, 85–88. National University of Singapore.

[2] Bevilacqua, F., Zamborlin, B., Sypniewski, A., Schnell, N., Guédy, F., & Rasamimanana, N. H. (2009). Continuous Realtime Gesture Following and Recognition. *Gesture Workshop*, *5934* (7), 73–84.

[3] Caramiaux, B., Wanderley, M. M., & Bevilacqua, F. (2012). Segmenting and Parsing Instrumentalists' Gestures. *Journal of New Music Research*, 41(1), 13–29.

[4] Carrillo, A. P., & Wanderley, M. M. (2012). Learning and extraction of violin instrumental controls from audio signal. *Mirum*, 25–30.

[5] Chang, C.-C., & Lin, C.-J. (2011). LIBSVM: A Library for Support Vector Machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3), 27.

[6] Collins, N. (2012). Automatic Composition of Electroacoustic Art Music Utilizing Machine Listening. *Computer Music Journal*, *36*(3), 8–23.

[7] Cont, A. (2006). Realtime Audio to Score Alignment for Polyphonic Music Instruments, using Sparse Non-Negative Constraints and Hierarchical HMMS. *IEEE International Conference on Acoustics, Speech and Signal Processing. Proceedings*, 5, V–V.

[8] Cont, A. (2008a). Antescofo: Anticipatory Synchronization and Control of Interactive Parameters in Computer Music. In *Proceedings of the International Computer Music Conference*, Ann Arbor.

[9] Cont, A. (2008b). *Modeling Musical Anticipation: From the Time of Music to the Music of Time*. ProQuest.

[10] Cont, A., Coduys, T., & Henry, C. (2004). Real-time Gesture Mapping in Pd Environment using Neural Networks. *Proceedings of NIME*, 39–42.

[11] Cont, A., Wessel, D., & Dubnov, S. (2014). Realtime Multiple-pitch and Multiple-instrument Recognition For Music Signals using Sparse Non-negative Constraints. In *Proceedings of Digital Audio Effects Conference*, Bordeaux, France.

[12] Fiebrink, R., & Cook, P. R. (2010). The Wekinator: a system for real-time, interactive machine learning in music. In *Proceedings of The Eleventh International Society for Music Information Retrieval Conference*. Utrecht.

[13] Fiebrink, R., Trueman, D., Britt, C., Nagai, M., Kaczmarek, K., Early, M., Daniel, M. R., Hege, A., and Cook, P. R. (2010) Toward understanding human-computer interaction in composing the instrument. In *Proceedings of the International Computer Music Conference*.

[14] Françoise, J. (2013). Gesture-sound mapping by demonstration in interactive music systems. *ACM Multimedia*, 1051–1054.

[15] Fujinaga, I., & MacMillan, K. (2000). Realtime recognition of orchestral instruments. In *Proceedings of the international computer music conference* (141), 43.

[16] Gillian, N., Knapp, B., & O'Modhrain, S. (2011). A Machine Learning Toolbox For Musician Computer Interaction. *Proceedings of the conference on New Interfaces for Musical Expression*, 343–348.

[17] Gillian, N., & Paradiso, J. A. (2014). The gesture recognition toolkit. *The Journal of Machine Learning Research*, *15*(1), 3483–3487.

[18] Grill, T. (2004). *flext — C++ programming layer for cross-platform development of PD and Max/MSP externals An introduction* In *Proceedings of The second Linux Audio Conference*.

[19] Hunt, A., & Thomas, D. (2000). *The pragmatic programmer: from journeyman to master*. Addison-Wesley Professional.

[20] Knapp, R. B. (2011). Recognition Of Multivariate Temporal Musical Gestures Using N-Dimensional Dynamic Time Warping. *Proceedings of the International Conference on New Interfaces for Musical Expression*, 1–6.

[21] Kohavi, R., & Provost, F. (1998). Glossary of terms. *Machine Learning*, *30*(2-3), 271–274.

[22] Malloch, J., Sinclair, S., & Wanderley, M. M. (2013). Libmapper: (a library for connecting things). *CHI Extended Abstracts*, 3087–3090.

[23] Momeni, A., & Wessel, D. (2003). Characterizing and Controlling Musical Material Intuitively with Geometric Models. *Proceedings of the conference on New Interfaces for Musical Expression*, 54–62.

[24] Nissen, S. (2003). Implementation of a fast artificial neural network library (fann). *Report*.

[25] Ono, M., Shizuki, B., & Tanaka, J. (2013). Touch & activate. Presented at the the 26th annual ACM symposium, New York, New York, USA: ACM Press, 31–40.

[26] Puckette, M. (1995). Formant-based audio synthesis using nonlinear distortion. *Journal of the Audio Engineering Society*, *43*(1), 40–47.

[27] Schmeder, A. W. (2004). Mapping Spectral Frames to Pitch with the Support Vector Machine [electronic resource].

[28] Smith, B. D., & Garnett, G. E. (2012). Unsupervised play: Machine learning toolkit for Max. *Proceedings of the conference on New Interfaces for Musical Expression.*

[29] Van Nort, D., Wanderley, M. M., & Depalle, P. (2014). Mapping Control Structures for Sound Synthesis: Functional and Topological Perspectives. *Computer Music Journal*, *38*(3), 6–22.

[30] Wessel, D., Freed, A., & Lee, M. (1991). Real-Time Neural Network Processing of Gestural and Acousti csignals. Presented at the International Computer Music Conference, 1–4.

[31] Zicarelli, D. (1991). Communicating with Meaningless Numbers. *Computer Music Journal*, *15*(4), 74–77.