Machine Learning for Makers: Interactive Sensor Data Classification Based on Augmented Code Examples

David A. Mellis†, Ben Zhang†, Audrey Leung‡, Björn Hartmann†
†: UC-Berkeley EECS, ‡: UC-Berkeley I School
mellis@berkeley.edu, benzh@cs.berkeley.edu, audrey@ischool.berkeley.edu, bjoern@berkeley.edu

ABSTRACT
Although many software libraries and hardware modules support reading data from sensors, makers of interactive systems often struggle to extract higher-level information from raw sensor data. Available general-purpose machine learning (ML) libraries remain difficult to use for non-experts. Prior research has sought to bridge this gap through domain-specific user interfaces for particular types of sensors or algorithms. Our ESP (Example-based Sensor Prediction) system introduces a more general approach in which interactive visualizations and control interfaces are dynamically generated from augmented code examples written by experts. ESP's augmented examples allow experts to write logic that guides makers through important steps such as sensor calibration, parameter tuning, and assessing signal quality and classification performance. Writing augmented examples requires additional effort. ESP leverages a fundamental dynamic of online communities: experts are often willing to invest such effort to teach and train novices. Thus support for particular sensing domains does not have to be hard-wired a priori by system authors, but can be provided later by its community of users. We illustrate ESP's flexibility by detailing pipelines for four distinct sensors and classification algorithms. We validated the usability and flexibility of our example-based approach through a one-day workshop with 11 participants.

Author Keywords
Maker culture, sensors, machine learning, examples

ACM Classification Keywords
H.5.2. User Interfaces: input devices and strategies; interaction styles; prototyping.

INTRODUCTION
Today's makers – whether hobbyists tinkering with electronics, designers prototyping new products, or educators guiding students in hands-on activities – have access to a wide range of sensors. Companies like SparkFun Electronics and Adafruit Industries make sensors available to hobbyists through the creation and distribution of breakout boards, software libraries, tutorials, and more. Platforms like Arduino and the Raspberry Pi facilitate writing code that interface with these sensors. In addition, a variety of toolkits provide novices with high-level hardware and software building blocks for working with sensors and other electronic components (e.g. [19, 25, 42, 49]). All of which makes it relatively straightforward for a hobbyist to purchase, connect, and get a reading from a sensor.

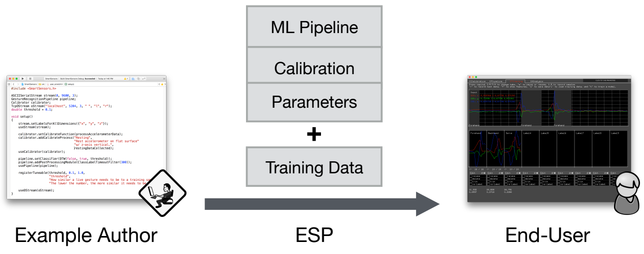
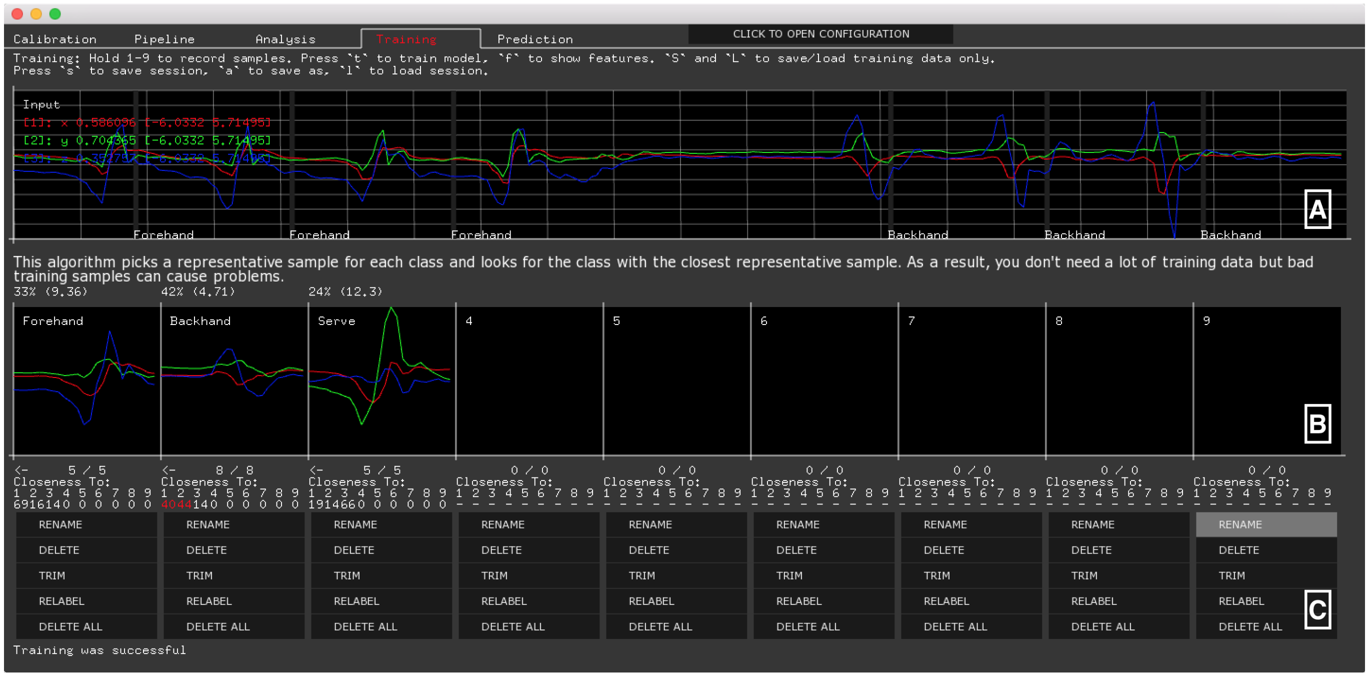


Figure 1. The ESP system leverages code examples written by machine learning experts to support end-users in applying ML to real-time, sensor-based applications.

It is, however, significantly more challenging to build interactive systems that analyze the data from these sensors in order to make higher-level inferences about physical phenomena — e.g. recognizing gestures from accelerometer data, or extracting information about the nature of a sound picked up by a microphone. Makers tend to use hard-coded thresholds for sensor readings, which is inadequate for real-world usage involving changing environmental and electrical conditions [33, 34].

At the same time, a vast array of signal processing and machine learning techniques have been developed by researchers and industry to detect a wide range of phenomena using sensor data. Powerful libraries such as SciKit-Learn [46] and MLPack [36] offer many sophisticated tools, but harnessing their power for a particular application domain requires significant machine learning expertise. As Domingos [11] has noted, much of the effort involved is in feature engineering – the selection of appropriate filters (*pre-processing modules*) and mathematical transformations (*feature extraction modules*) for transforming raw input data into a form suitable for processing by a machine learning classifier. Together with the

DIS 2017, June 10-14, 2017, Edinburgh, United Kingdom
ACM 978-1-4503-4922-2/17/06.
http://dx.doi.org/10.1145/3064663.3064735



```
Training: Hold 1-9 to record samples. Press `t` to train model. `f` to show features. `S` and `L` to save/load training data only.
Press `s` to save session. `a` to save as. `l` to load session.
```

```
[1]: x 0.586096 [-6.0332 5.71495]
[2]: y 0.704365 [-6.0332 5.71495]
```

This algorithm picks a representative sample for each class and looks for the class with the closest representative sample. As a result, you don't need a lot of training data but bad training samples can cause problems.

```
33% (9.36)          42% (4.71)          24% (12.3)

<-      5 / 5       <-      8 / 8       <-      5 / 5              0 / 0
Closeness To:       Closeness To:       Closeness To:       Closeness To:
1 2 3 4 5 6 7 8 9   1 2 3 4 5 6 7 8 9   1 2 3 4 5 6 7 8 9   1 2 3 4 5 6 7 8 9
6916140 0 0 0 0 0   4044140 0 0 0 0 0   1914660 0 0 0 0 0   - - - - - - - - -
```

```
Training was successful
```

Figure 2. The training tab of the ESP interface, running our gesture recognition example: (a) live sensor data with prediction from the machine learning pipeline, (b) training data with confusion scores ("closeness to"), (c) interface for editing training data.

classifier itself, and optional *post-processing modules* (e.g. for smoothing or de-bouncing classifier predictions), these pre-processing and feature extraction modules form a machine learning *pipeline*. Once a machine learning pipeline has been defined, it can often be applied to a variety of specific phenomena within the same general category, e.g. a gesture recognition pipeline can be applied to many different gestures, and a speaker identification pipeline can be applied to many different individual voices. This means that the hard work of feature engineering, once complete, can potentially be of use to many different projects. However, just providing a pipeline itself is not enough. Makers also need appropriate tools to manage the transfer and application of a pipeline to their specific project context, as well as methods to understand why a pipeline may not perform as intended, and how to improve this performance.

Prior HCI research has explored the potential of domain-specific interfaces for allowing ML novices to visualize and customize specific, fixed pipelines, e.g. for gesture recognition using dynamic time warping (MAGIC [3] and Exemplar [24]) and for image classification using decision trees (Crayons [12]). Developing a custom software application for each application domain does not scale, as significant effort is required even for machine learning experts already familiar with the domain. Other work provides generic interfaces for defining pipelines and managing training data [6, 13, 38], but assumes that the user is knowledgeable enough in machine learning to design and implement their own pipeline.

Our Example-based Sensor Prediction (ESP) system takes a community-centric approach to supporting makers in applying machine learning to a wide range of applications (Figure 1): expert community members author *augmented* code examples that define an ML pipeline, as well as additional logic to help novices customize this pipeline. When novices run these examples, ESP generates visualization and customization user interfaces based on these augmentations. Our approach is motivated by the tremendous success of online expertise sharing communities such as Stack Overflow [32] or the Arduino Forum [2], where expert users routinely invest efforts to create and share carefully crafted code examples and explanations. Thus support for particular sensing domains does not have to be hard-wired into ESP, but can be provided later by its community of users.

In ESP, experts author pipelines using a general-purpose machine learning library, the C++ Gesture Recognition Toolkit (GRT) [17]. This allows for diverse and complex pipelines and enables the expert to focus on machine learning, not GUI development. In addition to the pipeline itself, an expert-authored ESP example can include domain-specific heuristics for evaluating signal quality, calibrating the system to their specific sensor setups, and collecting quality training data. For instance, a gesture recognition example can support calibration for accelerometers with different ranges and provide warnings if the data is noisy. Experts can also provide domain-specific parameters for pipeline configuration, allowing for the translation of abstract machine learning terminology into a relevant and understandable vocabulary.

ESP generates a graphical interface from these augmentations (Figure 2) that allows makers to apply the pipelines to their own projects by iteratively modifying the training data and customizing pipeline parameters. ESP provides a range of pipeline-agnostic visualizations and operations inspired by prior research, supplying a common interface for novices to manage training data and to visualize and customize pipeline behavior. These elements include visualization of each stage of the pipeline, from raw sensor input to final features; numerical measures of the confusion between different classes of training data; and real-time plots of the distribution and confidence of pipeline predictions.

To ensure the flexibility of the ESP system, we developed it in parallel with a set of examples, motivated by techniques published in the HCI literature and encountered as aspirational goals in our classroom experience. These motivating examples involve different sensors and classifiers, using their diverse characteristics to inform the system's functionality and design. These examples are:

- gesture recognition using an accelerometer and dynamic-time warping (DTW)
- speaker identification using a microphone and a Gaussian Mixture Model (GMM) applied to MFCC coefficients
- grasp detection using swept-frequency capacitive sensing and a Support Vector Machine (SVM) (a la Touché [44])
- simple object recognition using a color sensor and a naive Bayes classifier

We describe some insights from a formative usability study with six participants that guided early development. We also describe insights from a full-day workshop with 11 participants, experienced makers with little knowledge of machine learning. This workshop validated the flexibility and usability of ESP's example-based approach, with participants finding a range of custom applications for the provided pipelines.

In summary, the contributions of this work are:

- Design of a system that supports experts in authoring augmented ML code examples, and novices in using and customizing such examples through auto-generated user interfaces for a wide range of interactive projects.
- Demonstration of the flexibility of such an example-based approach through the implementation of examples using a diversity of sensors and classifiers.
- Evaluation of the system through a hands-on workshop, highlighting the ways in which our examples-based system can be adopted by our target audience.

RELATED WORK

Here, we discuss three categories of related work: interfaces supporting the use of machine learning, example-centric approaches to supporting novices in other domains, and HCI applications of machine learning to sensing.

User Interfaces for Machine Learning

Prior interfaces for making machine learning more accessible often follow one of two general approaches. The first involves pairing a specific domain and a set of machine learning algorithms and then building an application and interface specifically tailored to them. This approach makes the system easier to use but limits its flexibility. For instance, the Crayons system [12] supports users in image classification through the use of decision trees and a pre-defined set of pixel-based features. PapierMache [27] and EyePatch [35] generalize this approach to support a wider variety of vision algorithms, but also supply specific UIs for specific classifiers. A CAPpella [10] supports end-users in building context-aware systems based on a pre-defined classes of video and audio events. Gesture Coder [31] learns to recognize multi-touch gestures from examples. Closer to our work are systems targeting real-time sensor data. MAGIC [3] is a system for gesture recognition using dynamic time warping with a fixed set of features derived from acceleration data. Similarly, Exemplar [24] supports the use of basic thresholding and dynamic time warping. Beyond these individual systems, Amershi's work [1] characterizes the larger design space for offering interactive control of machine learning systems.

A second approach is to generically support a wide range of machine learning algorithms and rely on the user to configure them for particular domains. This approach is more flexible but requires the user to decide which machine learning algorithms and features to use. Examples include Wekinator [13], Gestalt [38], and ml.lib [6]. A unique approach is taken by PICL [15], which turns machine learning into a hardware component that makers can add to their circuits.

Our approach is to build a generic system that supports many types of machine learning and to leverage expert-authored examples to scaffold novices for particular applications. Experts can embed a variety of domain-specific heuristics and enable ESP to visualize the behavior of the entire pipeline. The visualizations themselves are based on techniques introduced in the prior work above. Perhaps closest to our vision of this community-centric approach is the emerging practice of sharing Jupyter Notebooks in the data science community [26]. Such online notebooks can contain code and data, explanatory text, and visualizations. These notebooks focus on offline data analysis, and the choice of visualization is entirely left to the author. ESP formalizes scaffolding through the APIs it provides to authors, and focuses on real-time data classification.

Working with Examples

Examples can provide important scaffolding for learning, serve as inspiration, and lower the barrier to participation [30]. HCI research in programming and design tools has focused on facilitating the creation of more powerful examples [18, 37], improving searching and browsing of example corpora [4, 41] and enabling comparison between multiple available examples [28]. Other techniques focus on reusing existing web pages or visualizations as examples by extracting their underlying structure or data [29, 45]. We investigate the utility of examples in the novel domain of real-time machine learning, drawing on the insight that custom user interfaces can greatly extend the power of examples and the ability of novices to make use of them.

Machine Learning for Real-Time Sensor Data

HCI researchers have applied machine learning to a wide range of sensing-based applications. Examples include: applying electromyography to sense and classify muscle activations in the forearm to control games [43]; classifying capacitive touch patterns of electrodes on a tablet to infer how

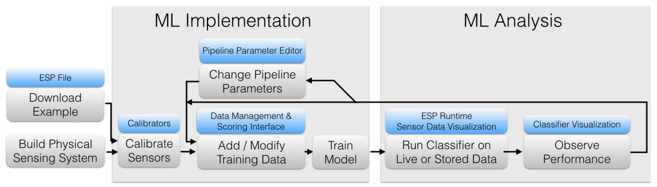

ML Implementation
ML Analysis
ESP File
Download Example
Build Physical Sensing System
Calibrators
Calibrate Sensors
Pipeline Parameter Editor
Change Pipeline Parameters
Data Management & Scoring Interface
Add / Modify Training Data
Train Model
ESP Runtime Sensor Data Visualization
Run Classifier on Live or Stored Data
Classifier Visualization
Observe Performance

Figure 3. Schematic overview of ESP : User actions are shown in gray; system support for those actions in blue.

it is being held [8]; recognizing which part of a user's finger hit a touch surface using acoustic features from a surface microphone [22]; and determining the location of finger taps on the body from microphone data [23]. Another set of ubiquitous computing systems aims to recognize activity in the home (e.g., faucet-level water usage, individual appliance usage) by deploying a single sensor on a house's infrastructure and applying machine learning [9, 14, 16, 20, 39]. We envision these projects as potential examples within our ESP system – that is, our system provides a framework for these researchers to share their work in a way that would facilitate its adoption and adaptation by novices.

THE ESP SYSTEM

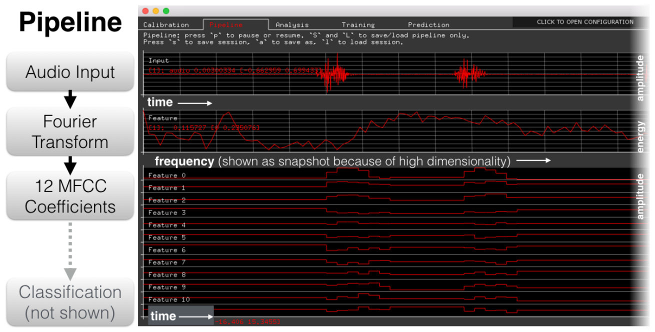
Our ESP system builds on expert-authored examples to provide an interface that allows machine-learning novices to modify and adapt machine-learning pipelines for use in their own projects. As Patel et al. [38] note, the process of applying machine learning typically involves frequent iteration between implementation and analysis. ESP supports machine-learning novices in this iterative process, as applied to the classification of real-time sensor data (Figure 3). With ESP, the pipeline itself is defined in the expert-authored example. Thus, for the ESP user, implementation means the collection and refinement of training data and the tuning of pipeline parameters. In particular, users need a way to easily record new training data from live streams of sensor data as well as an understandable vocabulary for the available parameters. For ESP, analysis means inspecting the behavior of the pipeline on incoming sensor data, observing the behavior of the overall interactive system containing the pipeline, and understanding the quality of and relationships between individual samples of training data. Here we discuss the interface of the ESP system (Figure 2) and the way it supports these processes of implementation and analysis for a machine-learning novice.

ESP is implemented in openFrameworks[1] and runs on Mac OS X, Windows, and Linux (Figure 7). ESP is open-source and available on GitHub.[2]

Visualizing Sensor Data

Machine learning pipelines may differ in the dimensionality of the sensor input, as well as the number and type of preprocessing and feature extraction stages. The ESP interface provides a pipeline-agnostic visualization of the live sensor data as it flows through the pipeline. We generalize the approach taken in MAGIC [3], showing live sensor data as a

[1]http://openframeworks.cc/
[2]https://github.com/damellis/esp



```
Pipeline: press `p` to pause or resume. `S` and `L` to save/load pipeline only.
Press `s` to save session, `a` to save as, `l` to load session.
[1]: audio 0.00300334 [-0.662959 0.699433]
[1]: 0.115727 [0 0.235076]
[-16.406 15.3455]
```

Figure 4. The pipeline tab of the ESP interface shows a live visualization of each stage of the machine learning pipeline. Here, we show the pipeline for our speaker identification example.

line graph, with each dimension in a different color. The interface allows the user to toggle this main live plot between the calibrated inputs from the sensor(s) and the final features being fed to the classification algorithm. By default, all ESP tabs show the live sensor data stream with an overlay of live predictions. That means that wherever the user is in the interface, they can always view the results of interacting with their sensor(s), facilitating experimentation. The pipeline tab visualizes each stage of the machine learning pipeline, graphing the sensor data resulting from each pre-processing and feature extraction module (Figure 4). To extract the required data, we query the pipeline using standard C++ interfaces – defined by the GRT and implemented by each module in the pipeline. For high dimensional data (above 32 dimensions), instead of a time-varying plot, we show a snapshot of the most recent data point.

Calibrating Sensors and Checking Signal Quality

Different users of ESP may have sensors or setups with different sensitivities or ranges, varying signal quality, and different physical setup. The ESP calibration tab allows the user to calibrate ESP for their particular setup and to receive feedback on their signal quality. These samples are analyzed by the expert-supplied code. For instance, our accelerometer-based example asks the user to collect a sample of data with the accelerometer upright and one with it upside-down. Our example code uses these samples to calculate the sensor values corresponding to 0 and 1g, as well as provide warnings in the presence of noise. Calibration allows the example and training data to be shared across accelerometers with different ranges. Expert-authored heuristics alert the user to potential signal quality issues that may interfere with the machine learning.

We also allow the expert to apply similar heuristics to each sample of training data collected by the user. This can provide domain-specific warnings, e.g. about the absence of motion in the recording of a gesture, or about noise in an audio sample.

Managing Training Data

A common aspect of the process of applying an existing machine learning pipeline to a particular project is the collection of project-specific training data. The ESP interface facilitates

Domain	ESP System	Example Author	ESP User
Machine Learning Pipeline	Visualizes pipeline stages, predictions, and likelihoods	Codes pipeline for a particular task	Incorporates pipeline into their project
Sensor Calibration	Provides interface for collection of calibration samples and applies calibration to incoming data	Specifies calibration samples required and code for processing them, as well as warnings about signal quality	Collects specified samples in GUI
Training Data Management	Provides interface for training data management. Calculates confusion scores between classes and information gain of new samples	Provides reference training data	Collects and modifies training samples for specific project
Pipeline Configuration	Provides interface for on-the-fly pipeline configuration	Specifies pipeline parameters that can be tuned	Tweaks the parameters based on their usage
Input and Output Specification	Provides a variety of input and output stream implementations	Selects and configures appropriate streams	Adjusts as necessary

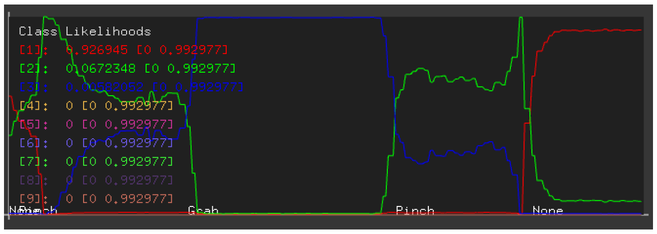
Table 1. Domains involved in applying machine learning to real-time sensor data and the work done by the ESP system, the example author, and the ESP user within each domain.

iterative collection and modification of training data and allows for on-the-fly re-training of the machine learning algorithm while it runs as part of the larger interactive project. The training tab (Figure 2) allows the user to collect, visualize, and edit training samples. Currently we support up to nine classes, which can be given custom names by the user. To collect new training samples, the user presses and hold the corresponding key on the keyboard ('1' to '9'). It's also possible to pause the pipeline and extract new training samples from the history of sensor readings. Furthermore, an expert can provide sample training data along with their example, providing the user with a starting point for experimentation and a visual reference of what good training data looks like. Because training data is stored as calibrated sensor data, it can be shared across physical setups and across pipelines with different pre-processing or feature-extraction modules.

Scoring Training Data

Our initial usability study made clear the importance of assisting users in understanding the quality of and relationship between individual training samples. To support this, we identified two scoring methods applicable to a wide range of machine learning pipelines: (1) confusion scores calculated on all samples when the model is trained and (2) information gain scores calculated on new training samples as they're collected.

Confusion scores can help users identify training samples which may be mislabeled or of insufficient quality to enable the machine learning pipeline to correctly identify their class. Specifically, whenever the user retrains the classifier, we score each training sample by running it through the pipeline as if it were live data, yielding the classifier's prediction of the likelihood of the sample belonging to each class of training data. (This approach has the advantage of not requiring any scoring-specific APIs in the underlying machine learning library.) Ideally, the model would predict with 100% likelihood that the sample belongs to the class to which the user has assigned it – with 0% probability of it belonging to any of the other classes. The lower the former number and the higher the latter, the more the sample is being confused with other classes, a sign that it may be mislabeled or of poor quality. We display these per-sample scores rather than an overall confusion matrix because it provides a direct connection to



```
Class Likelihoods
[1]:  0.926945 [0 0.992977]
[2]:  0.0672348 [0 0.992977]
[3]:  0.00582052 [0 0.992977]
[4]:  0 [0 0.992977]
[5]:  0 [0 0.992977]
[6]:  0 [0 0.992977]
[7]:  0 [0 0.992977]
[8]:  0 [0 0.992977]
[9]:  0 [0 0.992977]
```

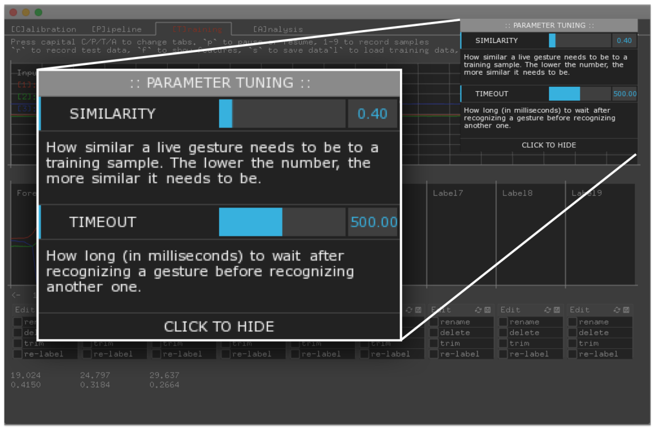
Figure 5. The prediction tab includes a plot of the relative likelihoods of each class of training data, shown here with the Touché example. The three lines correspond to the following classes: Red: No touch. Green: Pinch. Blue: Grab.

the per-sample editing operations the user might undertake as a result of the scores. Because the expected scores for good training samples vary across pipelines, we allow the expert to specify a custom threshold below which to display a warning about the corresponding training sample. This allows a little bit of work by the example author (specifying the threshold) to augment the value of the more complex work (score calculation) done by the system.

While confusion scores are applicable to a wide range of machine learning systems, ESP differs from the better-studied problems of data cleaning [5] or interactive labeling [47, 48] because our users can not only label or relabel existing training data but also collect new samples. In order to provide the user with information about new training samples as they're collected, we borrow an idea from Guyon et al. [21], who proposed an early online approach to data cleaning. Specifically, we calculate the information gain of each new sample by taking the negative log of the probability given by the classifier of the sample belonging to the class to which it was assigned by the user. This provides the user with a suggestion of the extent to which this new training sample will modify the behavior of the classifier. Scores with low information gain (e.g. 1-2%) suggest that the collection of additional training data is offering limited value, while scores with very high information gain (e.g. 90%+) may be a sign that the training sample has been misclassified by the user or is of poor quality.

Visualizing Classifier Results

If a machine learning pipeline isn't making the predictions its user expects, it can be helpful to view more details about the relationship between the live data and the various classes of



Press capital C/P/T/A to change tabs, `p` to pause or resume, 1-9 to record samples
`r` to record test data, `f` to show features, `s` to save data`l` to load training data,

:: PARAMETER TUNING ::

SIMILARITY 0.40

How similar a live gesture needs to be to a training sample. The lower the number, the more similar it needs to be.

TIMEOUT 500.00

How long (in milliseconds) to wait after recognizing a gesture before recognizing another one.

CLICK TO HIDE

Figure 6. The ESP interface allows the end-user to tune the parameters specified by the example author.

training data in order to find ways to improve the pipeline's performance. ESP's predictions tab (Figure 5) plots the class likelihoods over time – that is, for each class of training data, the probability of the live data belonging to that class. For models that support it, the predictions tab also plots the class distances – a model-specific notion of the "distance" from live data to each class. The class distance is typically used for null rejection: a prediction that the current live data does not belong to any of the trained classes – not even the one with the highest likelihood – because its distance from that class exceeds some threshold. This null rejection threshold often included in the expert-authored tuneable parameters as it allows the user to adjust the specificity of the predictions. To better explore these plots, the user can pause the ESP pipeline and mouse over the plots to view the numeric values for each moment in time.

Customizing Pipeline Parameters

Users may want to tune various aspects of a machine learning pipeline to their own needs, for instance to adjust false-positive vs. false-negative rates (i.e. precision and recall) or to customize "debouncing" intervals (i.e. a delay between successive predictions). In general, however, the code specifying a machine learning pipeline contains many such potential parameters – a large space likely to be overwhelming to those without machine learning expertise. We take an example-centric approach towards making the parameter space understandable to novices. The example author specifies, in code, parameters that can be tuned (Figure 8 (c) and (g)), along with a domain-specific name and description for each. The end-user can then, in the ESP GUI, view and adjust these parameters (Figure 6). These tunable parameters provide the end-user with a smaller, documented parameter space focused on their specific application, facilitating understanding and experimentation.

Validating Pipeline Correctness

As one edits a machine learning pipeline or its training data, a common practice is to periodically check the behavior of the pipeline on a consistent set of test or validation data. The analysis tab supports this practice, allowing the user to record

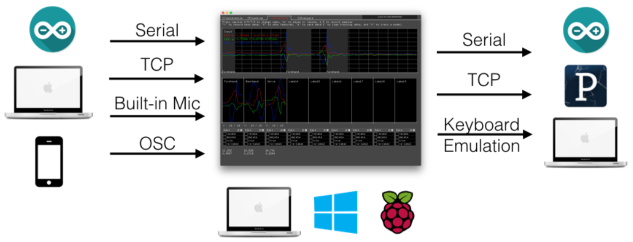


Figure 7. ESP can run within a user's larger interactive system, reading input and writing output to a variety of sources. The system can run on Mac, Windows, and Linux (including a preliminary port to the Raspberry Pi).

a longer sequence of data against which to test the pipeline's performance. The tab displays a visualization of the recorded test data with an overlay of the predictions made by the machine learning pipeline. The user can navigate through this visualisation by selecting a region of interest on a condensed view of the entire test data set. Whenever the machine learning pipeline is retrained (on updated training data or tuned parameters), prediction is re-run on the test data set and the plot updated.

Reading Sensor Data and Outputting Pipeline Results

Machine learning may be useful to add to interactive projects that employ a diversity of sensing and actuation methods. To support its use within these larger systems, ESP can read sensor data from a variety of input streams and output pipeline results to a variety of output streams. These include Arduino boards or other microcontrollers connected over USB serial, TCP network connections, and the built-in microphone on the computer ESP is running on. Only one input stream is supported but this can include readings from multiple sensors. ESP supports multiple simultaneous outputs, sending the same data to each. Because ESP continues to stream live data as the user adjusts the machine learning pipeline, they can explore and refine its behavior as part of their overall project.

ESP API FOR EXAMPLE AUTHORS

Here, we describe the API experts use to author an ESP example. (The full API documentation is included in the supplemental materials.) The pipeline itself is specified using the the Gesture Recognition Toolkit (GRT) [17], a general-purpose C++ library for real-time machine learning. We chose to build on the GRT because it is one of the only machine learning libraries specifically intended for use with live, timeseries data. In addition, the GRT's use of C++ offers the potential to port GRT pipelines to embedded platforms like Arduino or the Raspberry Pi – platforms that are frequently used by makers to extract data from sensors. We supply ESP-specific functions for specifying sensor calibration, pipeline parameter tuning, and input and output streams. We've attempted to keep the ESP API relatively similar in style to that of Arduino to facilitate modifications by novices if the example doesn't quite meet their needs.

```
   #include <ESP.h>

   ASCIISerialStream iStream(115200, 3);
   TcpOStream oStream("localhost", 5204);
   GestureRecognitionPipeline pipeline;
   Calibrator calibrator;

A  double zeroG = 0, oneG = 0;

   double processAccelerometerData(double input) {
     return (input - zeroG) / (oneG - zeroG);
   }

   CalibrateResult restingDataCollected(const MatrixDouble& data) {
       // take average of X and Y acceleration as the zero G value
       zeroG = (data.getMean()[0] + data.getMean()[1]) / 2;
       oneG = data.getMean()[2]; // use Z acceleration as one G value

       double range = abs(oneG - zeroG);
       vector<double> stddev = data.getStdDev();

       if (stddev[0] / range > 0.05 ||
           stddev[1] / range > 0.05 ||
           stddev[2] / range > 0.05)
           return CalibrateResult(CalibrateResult::WARNING,
               "Accelerometer readings are noisy, check circuit.");

       return CalibrateResult::SUCCESS;
   }

B  TrainingSampleCheckerResult checkTrainingSample(const MatrixDouble &in) {
       VectorDouble stddev = in.getStdDev();
       if (*max_element(stddev.begin(), stddev.end()) < 0.1)
           return TrainingSampleCheckerResult(TrainingSampleCheckerResult::WARNING,
               "Warning: Gesture contains very little movement.");
       return TrainingSampleCheckerResult::SUCCESS;
   }

C  double null_rej = 0.4;

   void updateVariability(double new_null_rej) {
       pipeline.getClassifier()->setNullRejectionCoeff(new_null_rej);
       pipeline.getClassifier()->recomputeNullRejectionThresholds();
   }

   void setup() {
D      useInputStream(iStream);
       useOutputStream(oStream);

E      calibrator.setCalibrateFunction(processAccelerometerData);
       calibrator.addCalibrateProcess("Resting",
           "Rest accelerometer on flat surface.", restingDataCollected);
       useCalibrator(calibrator);

F      DTW dtw(false, true, null_rej);

       pipeline.setClassifier(dtw);
       usePipeline(pipeline);

G      registerTuneable(null_rej, 0.1, 5.0, "Variability",
            "How different from the training data a new gesture can be and "
            "still be considered the same gesture. The higher the number, the "
            "more different it can be.", updateVariability);

H      useTrainingSampleChecker(checkTrainingSample);
   }
```

Figure 8. Code for creating a pipeline for a gesture recognition system: (a) code for performing calibration (the `processAccelerometerData()` function normalizes each sensor reading, the `restingDataCollected()` function calculates the calibration parameters based on the user's sample), (b) function for warning users if training samples contain little motion, (c) variable and function for a user-tuneable parameters, (d) registration of input and output streams, (e) specifying the calibration samples and functions used, (f) definition of the machine learning pipeline, (g) definition of the tuneable parameter, (h) registration of the training sample checker. Note that for space, we show a slightly simpler version of our gesture recognition example than the one described in the paper.

Machine Learning Pipeline

The GRT includes a large and extensible set of modules for the creation of machine learning pipelines, including classifiers, pre-processing filters, feature extraction modules, and post-processing filters, each implementing standard base classes. This allows experts to leverage tested implementations of common modules while enabling the creation of custom modules whose interfaces are compatible with the GRT and ESP. For instance, we implemented a feature-extraction module to derive MFCC coefficients from an FFT spectrum for our speaker identification example.

Calibration Specifications

In an ESP example, calibration is specified by one or more calibration samples and a mapping function. An example is shown in Figure 8(a) and (e). Each calibration sample has a string name and description that is shown to the user. The example author provides a callback function to process each of these samples as they're collected by the user, extracting any required information (e.g. mean, standard deviation) from the recorded series of data points. The callback can also signal a warning or error and provide a message to be shown to the user. In case of error, the calibration sample is rejected and the user must collect a new one. After all the calibration samples have been recorded, live data is passed to the specified mapping function, which translates each sensor reading from raw input values (e.g. ADC readings) to calibrated values (e.g. in units of g-force).

Training Sample Heuristics

To provide domain-specific heuristics on the user's training data, example authors can register a function to be called on each training sample as it's recorded (Figure 8(b) and (h)). This function analyzes the recorded sensor readings and returns either success, or a warning or error together with an associated message to be displayed to the user. For instance, our gesture recognition example warns the user about unusually short or still gestures.

Pipeline Tuning

Tunable parameters are specified as a reference to a global variable (integer, floating point, or boolean), a default value, a minimum and maximum value (for numeric types), a name, a description, and a function to be called when the parameter's value is changed by the user (Figure 8 (c) and (g)). This callback will typically update one or more parameters in the underlying machine learning pipeline, possibly translating to standard units from domain-specific values.

Input and Output Data Streams

ESP examples can instantiate and register any of a number of input, output, and bi-directional data streams. ESP registers a callback with the registered input stream; the callback is supplied with incoming data points by the input stream. ESP calls any registered output streams with the output of the active pipeline (i.e. the number of the predicted class).

SAMPLE ESP APPLICATIONS

Here, we describe the application of our system to four machine learning problems as a means of illustrating the flexibility of our example-centric approach (see Table 2). We also use these applications to describe how the features of ESP enabled us to improve recognition accuracy.

Application 1. Gesture Recognition

This example shows how, by writing a few lines of code in our system, an expert can provide end-users with an interface similar to that of specialized gesture recognition tools like Exemplar or MAGIC. A simplified version of the example code is shown in Figure 8. It instantiates a machine learning pipeline with a dynamic time warping classifier provided by the GRT. This pipeline is fed by an ESP input stream that receives accelerometer values from an Arduino board connected via USB serial. We've included a calibrator that allows

Example	Input	Calibration	Features	Classifier	Parameters
Gesture Recognition	Accelerometer	Bias & Sensitivity	Raw XYZ	DTW	Similarity & Timeout
"Touché"	Electrode	–	Frequency response	SVM	–
Speaker Identification	Microphone	Bias & Sensitivity	MFCC	GMM	Noise level
Color	Color Sensor	–	Raw RGB	Naive Bayes	Scaling & Similarity

Table 2. ESP examples.

the code to be used with accelerometers of differing ranges. We also specified two parameters that can be tuned by the end-user: one that controls the strictness of the recognition and another (not pictured) that controls the minimum delay between successive predictions of a gesture. With this example, ESP users can train the system to recognize a wide range of non-repetitive gestures (e.g. tennis swings, dance steps, etc). In applying this example to the recognition of different gestures, we found the confusion scores valuable in suggesting training samples to delete and the class distance plots essential to tuning the classifier's null rejection threshold.

Application 2. Swept-Frequency Capacitive Sensing

This example provides Touché [44]-like functionality, using swept-frequency capacitive sensing to detect different user grasps or poses. We modified existing Arduino code[3] to generate the variable frequencies, read the corresponding capacitance values, and stream them to the computer. This generates a 160-dimensional input vector, which we supply to an SVM (w/ polynomial kernel). (For convenience, we slightly simplified the pipeline used in the original publication.) This example challenged us to develop visualizations and interactions appropriate to such high-dimensional data streams. It allows users to detect interactions with a range of capacitive surfaces (e.g. different ways of grasping a water bottle, or of touching two hands together). In applying this example to different uses, the class likelihood plot was helpful in clarifying the physical positions corresponding to the boundaries between predicted classes and the extent to which the classifier was able to distinguish between different physical configurations.

Application 3. Speaker Identification

The speaker identification example determines the speaker from a known group of voices. It uses Mel-Frequency Cepstral Coefficients (MFCCs) as the features and Gaussian Mixture Models (GMM) as the classifier [40]. Two types of input audio streams are supported: a computer's built-in microphone or an external microphone connected via an Arduino board. Because the GRT doesn't include a MFCC feature extraction module, we implemented our own (following the HTK book [50]). We use 30 millisecond audio frames (with 15 ms overlap). Each frame generates 12 MFCC coefficients, the feature vector fed into the 16-mixture GMM classifier.

We learned several lessons from building this example. (1) Processing high-rate audio data is computational expensive. Our initial MFCC implementation follows the reference equations without much optimization. It has slowed down the whole user interface, making the system un-usable. To speed up, we optimized the code by reducing object construction whenever possible and rely on BLAS[4] for the actual matrix computation. (2) Training GMM models is time-consuming, typically on the order of minutes. The training is also not guaranteed to succeed because of the underlying EM (expectation-maximization) algorithm might not converge. Our pipeline saving and loading functionality played an important role in developing this example.

Application 4. Color Sensing

This example classifies objects based on readings from a color sensor. While this simple example could potentially be implemented without a machine learning algorithm, our interface facilitates the collection and testing of sensor readings. Our feature vector consists of the red, green, and blue values from the color sensor, normalized to unit length. We feed these values to a naive Bayes classifier. With this application, the information gain scores proved useful in understanding when we could stop collecting additional training data because it doing little to update the behavior of the classifier.

EVALUATION

We evaluated ESP through a one-day workshop with 11 participants. Here, we describe the workshop participants and structure, then discuss insights it offers into the ESP system.

Workshop Participants and Structure

The workshop took place in a maker space on our campus and lasted from 10 am to 6 pm on a Saturday. Participants completed pre- and post-workshop online surveys and participated in opening, mid-workshop, and closing discussions (all audio-recorded). We recruited participants who were experienced with physical computing but relative novices at machine learning (as self-assessed on an initial screening survey). Of our 11 participants, three were female and eight male. They ranged in age from 25 to 61 (median 39). Six participants have a master's degree or PhD. On the opening survey, six reported having no prior experience with machine learning, two reported a little experience, and two reported some experience. (One participant arrived late and didn't complete the opening survey.) Eight of the ten participants who completed the opening survey reported significant experience in either programming, electronics, or sensors, with the remaining two reporting at least some experience in one of the areas.

The workshop was facilitated by the two first authors, with assistance from the main author of the GRT. After we briefly introduced the ESP interface, participants spent about an hour exploring the recognition of custom gestures using our accelerometer- and DTW-based example. Afterwards, they

[3]http://www.instructables.com/id/Touche-for-Arduino-Advanced-touch-sensing/

[4]http://www.netlib.org/blas/

each shared reflections on the process. We then provided a brief overview of the dynamic time warping algorithm used in the example. After lunch, we introduced the Touché and color sensing examples, after which participants experimented with the recognition of custom classes of training data. This was followed by participant reflections and a short presentation of the SVM and naive Bayes algorithms in the examples. Finally, participants had about an hour to experiment freely – with some exploring additional classes of training data for the provided examples, others connecting ESP prediction output back to Arduino or to Processing, and some modifying ESP example code.

Discussion

Here, we discuss four themes that emerged from participants' use of ESP in the workshop. Participant feedback also suggested several directions for future research, which we discuss in the following section ("Limitations and Future Work").

Provided Insight into Real-World Use of Machine Learning

Using the ESP system gave our participants insight into the application of machine learning to real-world projects. At a basic level, it gave them the confidence that they could do so. This is evidenced by their survey responses – the number of participants agreeing or strongly agreeing with the statement "I could incorporate machine learning into an interactive project if I wanted to" went from one before the workshop to nine after it.

The example-based nature of ESP allowed participants to experiment with a variety of sensors and machine learning algorithms:

"Getting multiple sensors (accelerometer, sound and color) to work was exciting and revealed [a lot] of possibility." (P9, closing survey)

"today i saw a bunch of really great specific examples of applying different classifiers and filters to different kinds of sensor data streams for different purposes. i feel that i [now] have a foundation to build on for investigating future projects." (P6, closing survey)

By using ESP, these participants learned about machine learning as a general strategy that can be applied to a range of real-time sensing tasks – in contrast to single-purpose tools like MAGIC, which expose users to a specific task like gesture recognition.

One participant contrasted their experience in the workshop with prior machine learning exercises involving canned training data and pre-determined outcome:

"This was totally real and raw and I was generating the data but then I was training model and seeing how well the classifier worked. And that whole process for me of being really hands-on with it really helped my way of thinking about it, thinking about all the steps along the way." (P5, closing discussion)

By experimenting with the use of ESP examples to recognize custom physical phenomena, these participants gained insight into the importance of quality training data for the application of machine learning. By going through the process of applying machine learning to their own uses, they learned that it is not a magical process but a specific set of techniques and processes, requiring iteration and understanding.

Demonstrated Successful Transfer of Provided Examples to Users' Tasks

Participants were able to experiment with the application of pre-existing machine learning pipelines to a range of sensing tasks. One pair of participants (P6 and P11) applied the Touché example to multiple tasks: distinguishing between the two of them based on their grasp of a soda can, detecting different hand gestures using an electrode wrapped around the wrist, and distinguishing different grasps on a water bottle. Another participant (P10) mounted their accelerometer to a pencil and used the DTW example to detect different types of scribbles. This suggests that ESP's example-based approach is flexible enough to allow machine-learning novices to apply an existing expert-authored pipeline to their own training data and projects.

Several of the participants identified ways in which they might use ESP in their own projects. P7 mentioned an installation that involved detecting people's proximity to a crystal ball. P6 was interested in building public art that involved detection of people's interaction with everyday objects. Both saw possibilities for implementing these using custom training data together with the ESP Touché example. P9 was interested in distinguishing between different colors of sky and explored the sensitivity of the color detection example for this purpose. This suggests that they see enough flexibility in ESP's examples to find them applicable to their real-world interests and projects.

Suggested the Accessibility of Example Modification

Two participants (P5 and P7) went beyond the workshop structure and modified one of the provided examples to work with their own sensor – a light sensor (P5) and a simple capacitive electrode (P7).

"I found it pretty easy to change the code and actually get something else working with another sensor.... I was able to successfully get something working all the way through to Processing and could change the screen graphics based on something I trained.... And that only took me, I don't know, 15 minutes to do." (P5, closing discussion)

"It was pretty easy to hook something completely new up to it, so I wouldn't worry about crossing that border." (P7, closing discussion)

Notably, this occurred without a section of the workshop dedicated to explaining or encouraging the editing of the ESP example code. This suggests that experienced Arduino users like these will be comfortable diving into and editing the code of ESP examples. This offers a path for ESP users to begin to experiment with authoring machine learning pipelines for themselves.

Provided Feedback on ESP Heuristics

Participant feedback offered some general insights into the value of the heuristics in ESP. Calibration, required for the gesture recognition example at the start of the workshop, proceeded without difficulty for all participants. The confusion scores were not immediately understandable to all participants, which prompted us to add the expert-supplied threshold for displaying warnings on problematic samples. This provides first pass indication of the quality of the training samples and may help users begin to explore the confusion scores in more detail. Participants' reflection on the gesture recognition example suggested opportunities for future training sample checkers, such as providing warnings for overly long or repetitive samples. Participants also suggested UI improvements for making the plots of the classifier results more legible.

LIMITATIONS

The current version ESP has several important limitations—some are conceptual, others due to implementation choices.

Only Supports Classification: ESP currently only supports multi-class classification problems. Machine learning is also used for other tasks such as regression or real-time gesture following [7]. Supporting such tasks will require different ways to assess the performance of a pipeline in ESP.

Focus on "Small Data:" The interactions in ESP are optimized for interactive, real-time training and classification using a small number of training examples and features. This restricts the applicability of the interfaces. Many recent advances in machine learning use orders of magnitude more data and features, where individual review of features and training examples becomes infeasible. Such approaches will require entirely different visualizations. We posit that the space of "small data" ML is still rich enough for many applications, and remains underserved.

Cost of Generality: ESP aims to support a broader range of application scenarios than prior systems such as MAGIC [3] or Exemplar [24]. Ideally, ESP authors should thus be able to provide re-implementations of these tools by writing appropriate ESP examples. However, to limit the work for authors, our API also seeks to insulate example authors from UI details. This abstraction prevents example authors from implementing advanced interaction techniques, such as Exemplar's visual interpretation of double thresholds. The tradeoff that limiting effort for example authors also limits expressivity is fundamental to our approach.

No Embedded Runtime: While ESP focuses on maker projects, which frequently focus on embedded microcontrollers, our current implementation still requires a desktop PC to run the real-time classification. We started an effort to port ESP to embedded systems, including initial cross-compilation for the Raspberry Pi and the execution of trained pipelines on embedded processors (e.g. ARM Cortex M4).

Only Evaluated Experience of Example Users: A core contribution of our work is to shift support of specific application scenarios from ESP system authors to expert community members who create examples. While we have informally discussed our approach with several educators and example authors in the Arduino community, we have not yet formally evaluated whether the API we provide is expressive and complete enough for authors.

FUTURE WORK

Here we discuss two larger areas for future work on scaffolding novice use of machine learning.

Supporting the Definition of Machine Learning Pipelines

As we noted in the introduction, the process of feature engineering is often one of the most difficult aspects of applying machine learning to a particular domain. As a result, in our work thus far, we chose not to focus on the problem of helping non-expert users author machine learning pipelines from scratch, instead relying on experts to encode them in examples. Supporting machine learning novices in the process of pipeline definition seems ripe for future research, whether through the exploration of automated approaches to feature selection, interfaces to help users define appropriate features for themselves, or repositories of pipelines showcasing various approaches.

Pipeline Composition

Feedback from our workshop participants suggested the importance of applying multiple machine learning approaches with an application domain. Within the general area of gesture recognition, participants mentioned recognition of periodic gestures, extracting the speed at which a gesture was performed, and orientation-independent recognition – all of which likely require different pipelines. This suggests the importance of exploring interfaces for the composition or comparison of multiple pipelines.

CONCLUSION

In this paper, we've shown how an example-centric interface can support machine-learning novices in applying machine learning to a wide range of real-time sensor-based applications. With our ESP system, machine-learning experts author code examples that encode a classification pipeline for a particular application domain, together with domain-specific heuristics. The ESP interface provides users with pipeline-agnostic visualizations and operations for the iterative collection of training data and pipeline customization. We highlighted the flexibility of ESP through the description of our development of four sample applications across different sensors and machine learning classifiers. The results of our one-day workshop suggested that ESP allows machine learning novices to apply and adapt expert-authored pipelines to their own uses. Finally, we suggested potential future research directions. In short, we feel that an example-based approach has great potential to make machine learning accessible to makers of a wide range of interactive systems.

ACKNOWLEDGEMENTS

This work was supported in part by TerraSwarm, one of six centers of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA, and by NSF CNS 1505773 and IIS 1149799. Thanks to Nick Gillian and the workshop participants!

REFERENCES

1. Amershi, S. *Designing for Effective End-User Interaction with Machine Learning*. PhD thesis, University of Washington, 2012.
2. Arduino Forum. `https://forum.arduino.cc/`, 2017. [Online; accessed 16-January-2017].
3. Ashbrook, D., and Starner, T. Magic: A motion gesture design tool. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI ’10, ACM (New York, NY, USA, 2010), 2159–2168.
4. Brandt, J., Dontcheva, M., Weskamp, M., and Klemmer, S. R. Example-centric programming: Integrating web search into the development environment. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI ’10, ACM (New York, NY, USA, 2010), 513–522.
5. Brodley, C. E., and Friedl, M. A. Identifying mislabeled training data. *Journal of Artificial Intelligence Research 11* (1999), 131–167.
6. Bullock, J., and Momeni, A. ml.lib: Robust, cross-platform, open-source machine learning for max and pure data. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, NIME ’15 (2015), 265–270.
7. Caramiaux, B., Montecchio, N., Tanaka, A., and Bevilacqua, F. Adaptive gesture recognition with variation estimation for interactive systems. *ACM Trans. Interact. Intell. Syst. 4*, 4 (Dec. 2014), 18:1–18:34.
8. Cheng, L.-P., Hsiao, F.-I., Liu, Y.-T., and Chen, M. Y. irotate grasp: Automatic screen rotation based on grasp of mobile devices. In *Adjunct Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology*, UIST Adjunct Proceedings ’12, ACM (New York, NY, USA, 2012), 15–16.
9. Cohn, G., Gupta, S., Froehlich, J., Larson, E., and Patel, S. N. Gassense: Appliance-level, single-point sensing of gas activity in the home. In *Proceedings of the 8th International Conference on Pervasive Computing*, Pervasive’10, Springer-Verlag (Berlin, Heidelberg, 2010), 265–282.
10. Dey, A. K., Hamid, R., Beckmann, C., Li, I., and Hsu, D. A cappella: Programming by demonstration of context-aware applications. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI ’04, ACM (New York, NY, USA, 2004), 33–40.
11. Domingos, P. A few useful things to know about machine learning. *Commun. ACM 55*, 10 (Oct. 2012), 78–87.
12. Fails, J., and Olsen, D. A design tool for camera-based interaction. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI ’03, ACM (New York, NY, USA, 2003), 449–456.
13. Fiebrink, R., Trueman, D., and Cook, P. R. A meta-instrument for interactive, on-the-fly machine learning. In *NIME* (2009), 280–285.
14. Fogarty, J., Au, C., and Hudson, S. E. Sensing from the basement: A feasibility study of unobtrusive and low-cost home activity recognition. In *Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology*, UIST ’06, ACM (New York, NY, USA, 2006), 91–100.
15. Fourney, A., and Terry, M. Picl: Portable in-circuit learner. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology*, UIST ’12, ACM (New York, NY, USA, 2012), 569–578.
16. Froehlich, J. E., Larson, E., Campbell, T., Haggerty, C., Fogarty, J., and Patel, S. N. Hydrosense: Infrastructure-mediated single-point sensing of whole-home water activity. In *Proceedings of the 11th International Conference on Ubiquitous Computing*, UbiComp ’09, ACM (New York, NY, USA, 2009), 235–244.
17. Gillian, N., and Paradiso, J. A. The gesture recognition toolkit. *J. Mach. Learn. Res. 15*, 1 (Jan. 2014), 3483–3487.
18. Ginosar, S., De Pombo, L. F., Agrawala, M., and Hartmann, B. Authoring multi-stage code examples with editable code histories. In *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology*, UIST ’13, ACM (New York, NY, USA, 2013), 485–494.
19. Greenberg, S., and Fitchett, C. Phidgets: Easy development of physical interfaces through physical widgets. In *Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technology*, UIST ’01, ACM (New York, NY, USA, 2001), 209–218.
20. Gupta, S., Reynolds, M. S., and Patel, S. N. Electrisense: Single-point sensing using emi for electrical event detection and classification in the home. In *Proceedings of the 12th ACM International Conference on Ubiquitous Computing*, UbiComp ’10, ACM (New York, NY, USA, 2010), 139–148.
21. Guyon, I., Matic, N., and Vapnik, V. Discovering informative patterns and data cleaning. In *AAAI Workshop on Knowledge Discovery in Databases* (1994).
22. Harrison, C., Schwarz, J., and Hudson, S. E. Tapsense: Enhancing finger interaction on touch surfaces. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST ’11, ACM (New York, NY, USA, 2011), 627–636.
23. Harrison, C., Tan, D., and Morris, D. Skinput: Appropriating the body as an input surface. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI ’10, ACM (New York, NY, USA, 2010), 453–462.

24. Hartmann, B., Abdulla, L., Mittal, M., and Klemmer, S. R. Authoring sensor-based interactions by demonstration with direct manipulation and pattern recognition. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '07, ACM (New York, NY, USA, 2007), 145–154.
25. Hartmann, B., Klemmer, S. R., Bernstein, M., Abdulla, L., Burr, B., Robinson-Mosher, A., and Gee, J. Reflective physical prototyping through integrated design, test, and analysis. In *Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology*, UIST '06, ACM (New York, NY, USA, 2006), 299–308.
26. Project Jupyter. `http://jupyter.org/`, 2017. [Online; accessed 16-January-2017].
27. Klemmer, S. R., Li, J., Lin, J., and Landay, J. A. Papier-mache: Toolkit support for tangible input. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '04, ACM (New York, NY, USA, 2004), 399–406.
28. Kong, N., Grossman, T., Hartmann, B., Agrawala, M., and Fitzmaurice, G. Delta: A tool for representing and comparing workflows. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, ACM (New York, NY, USA, 2012), 1027–1036.
29. Kumar, R., Talton, J. O., Ahmad, S., and Klemmer, S. R. Bricolage: Example-based retargeting for web design. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, ACM (New York, NY, USA, 2011), 2197–2206.
30. Lee, B., Srivastava, S., Kumar, R., Brafman, R., and Klemmer, S. R. Designing with interactive example galleries. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, ACM (New York, NY, USA, 2010), 2257–2266.
31. Lü, H., and Li, Y. Gesture coder: A tool for programming multi-touch gestures by demonstration. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, ACM (New York, NY, USA, 2012), 2875–2884.
32. Mamykina, L., Manoim, B., Mittal, M., Hripcsak, G., and Hartmann, B. Design lessons from the fastest q&#38;a site in the west. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, ACM (New York, NY, USA, 2011), 2857–2866.
33. Martin, F. G. Ideal and real systems: A study of notions of control in undergraduates who design robots. In *In Y. Kafai and M. Resnick (Eds.), Constructionism in Practice: Rethinking the Roles of Technology in Learning*, Citeseer (1996).
34. Martin, F. G. Real robots don't drive straight. In *AAAI Spring Symposium: Semantic Scientific Knowledge Integration* (2007), 90–94.
35. Maynes-Aminzade, D., Winograd, T., and Igarashi, T. Eyepatch: Prototyping camera-based interaction through examples. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology*, UIST '07, ACM (New York, NY, USA, 2007), 33–42.
36. mlpack C++ machine learning library. `http://www.mlpack.org/`, 2017. [Online; accessed 16-January-2017].
37. Oney, S., and Brandt, J. Codelets: Linking interactive documentation and example code in the editor. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, ACM (New York, NY, USA, 2012), 2697–2706.
38. Patel, K., Bancroft, N., Drucker, S. M., Fogarty, J., Ko, A. J., and Landay, J. Gestalt: Integrated support for implementation and analysis in machine learning. In *Proceedings of the 23Nd Annual ACM Symposium on User Interface Software and Technology*, UIST '10, ACM (New York, NY, USA, 2010), 37–46.
39. Patel, S. N., Robertson, T., Kientz, J. A., Reynolds, M. S., and Abowd, G. D. At the flick of a switch: Detecting and classifying unique electrical events on the residential power line. In *Proceedings of the 9th International Conference on Ubiquitous Computing*, UbiComp '07, Springer-Verlag (Berlin, Heidelberg, 2007), 271–288.
40. Reynolds, D. A. Speaker identification and verification using gaussian mixture speaker models. *Speech Commun. 17*, 1-2 (Aug. 1995), 91–108.
41. Ritchie, D., Kejriwal, A. A., and Klemmer, S. R. D.tour: Style-based exploration of design example galleries. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, ACM (New York, NY, USA, 2011), 165–174.
42. Sadler, J., Durfee, K., Shluzas, L., and Blikstein, P. Bloctopus: A novice modular sensor system for playful prototyping. In *Proceedings of the Ninth International Conference on Tangible, Embedded, and Embodied Interaction*, TEI '15, ACM (New York, NY, USA, 2015), 347–354.
43. Saponas, T. S., Tan, D. S., Morris, D., Balakrishnan, R., Turner, J., and Landay, J. A. Enabling always-available input with muscle-computer interfaces. In *Proceedings of the 22Nd Annual ACM Symposium on User Interface Software and Technology*, UIST '09, ACM (New York, NY, USA, 2009), 167–176.
44. Sato, M., Poupyrev, I., and Harrison, C. Touché: Enhancing touch interaction on humans, screens, liquids, and everyday objects. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, ACM (New York, NY, USA, 2012), 483–492.

45. Savva, M., Kong, N., Chhajta, A., Fei-Fei, L., Agrawala, M., and Heer, J. Revision: Automated classification, analysis and redesign of chart images. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, ACM (New York, NY, USA, 2011), 393–402.

46. scikit learn: Machine Learning in Python. `http://scikit-learn.org/`, 2017. [Online; accessed 16-January-2017].

47. Tong, S., and Chang, E. Support vector machine active learning for image retrieval. In *Proceedings of the Ninth ACM International Conference on Multimedia*, MULTIMEDIA '01, ACM (New York, NY, USA, 2001), 107–118.

48. Tong, S., and Koller, D. Support vector machine active learning with applications to text classification. *Journal of machine learning research 2*, Nov (2001), 45–66.

49. Villar, N., Scott, J., Hodges, S., Hammil, K., and Miller, C. .net gadgeteer: A platform for custom devices. In *Proceedings of the 10th International Conference on Pervasive Computing*, Pervasive'12, Springer-Verlag (Berlin, Heidelberg, 2012), 216–233.

50. Young, S., Evermann, G., Gales, M., Hain, T., Kershaw, D., Liu, X., Moore, G., Odell, J., Ollason, D., Povey, D., et al. The htk book. *Cambridge university engineering department 3* (2002), 175.