Machine Learning Education for Artists, Musicians, and Other Creative Practitioners

REBECCA FIEBRINK, Department of Computing, Goldsmiths University of London

This article aims to lay a foundation for the research and practice of machine learning education for creative practitioners. It begins by arguing that it is important to teach machine learning to creative practitioners and to conduct research about this teaching, drawing on related work in creative machine learning, creative computing education, and machine learning education. It then draws on research about design processes in engineering and creative practice to motivate a set of learning objectives for students who wish to design new creative artifacts with machine learning. The article then draws on education research and knowledge of creative computing practices to propose a set of teaching strategies that can be used to support creative computing students in achieving these objectives. Explanations of these strategies are accompanied by concrete descriptions of how they have been employed to develop new lectures and activities, and to design new experiential learning and scaffolding technologies, for teaching some of the first courses in the world focused on teaching machine learning to creative practitioners. The article subsequently draws on data collected from these courses—an online course as well as undergraduate and masters-level courses taught at a university—to begin to understand how this curriculum supported student learning, to understand learners' challenges and mistakes, and to inform future teaching and research.

CCS Concepts: • **Social and professional topics → Computing education**; • **Computing methodologies → Machine learning**; • **Applied computing → Arts and humanities**;

Additional Key Words and Phrases: Machine learning education, Creative computing, STEAM

ACM Reference Format:
Rebecca Fiebrink. 2019. Machine Learning Education for Artists, Musicians, and Other Creative Practitioners. *ACM Trans. Comput. Educ.* To appear March 2019, 32 pages. https://doi.org/10.1145/3294008

1 INTRODUCTION

Machine learning (ML) systems that autonomously create new works of art, music, video, or text—some that reasonably mimic creators (e.g., [9], [14]), others that create content imbued with a novel machine "style" [52]—have recently captured significant public interest. But ML has in fact been used for many years in a wide range of human creative practices. These include building new digital musical instruments controlled by sensors [47], creating computer-based "accompanists" for human acoustic musicians [61], tracking and responding to dancers' movements [12], and building interactive art influenced by data such as audience actions or emotional states [39].

Historically, applying ML in creative work—as in other domains—has required deep knowledge of ML and programming. While some recent software can enable people without extensive programming expertise to use ML in new creative projects (e.g., [10, 25, 29]), these tools still require some familiarity with ML—for instance, to "debug" an ML system that is not working, and to understand when ML is likely to be useful. But there is little research about teaching ML to non-computer-scientists, and especially little work examining how or what to teach creative practitioners about ML. Indeed, there is little evidence of any courses about ML being designed for creative practitioners prior to the MOOC ("massively open online class") described in this paper, which began in 2016.

Author's address: Rebecca Fiebrink, Department of Computing, Goldsmiths University of London, r.fiebrink@gold.ac.uk.

Still, providing appropriate ML education and tools to creative practitioners presents many potential benefits: by enabling more people to use ML more effectively, ML education can lead to new creative outputs and means for self-expression, and also to economic impact as creative entrepreneurs discover opportunities for using ML in creative technologies (e.g., at least one MOOC student has started a company making creative ML technology [60]). By better understanding how to teach creative practitioners about ML, we can also inform the teaching of ML to other groups.

As early work on this subject, this article thus aims to contribute an inaugural understanding of why, what, and how to teach creative practitioners about ML. Section 2 begins with an overview of the uses of ML in creative work, of creative computing education, and of ML education research. Section 3 then draws on research about design processes in engineering and creative practice to motivate a set of learning objectives for students who wish to design new creative artifacts with ML. Section 4 then draws on education research and knowledge of creative computing practices and practitioners to argue for a set of teaching strategies that can be used to support these objectives, particularly for students doing creative work who may lack strong programming and/or math backgrounds. Examples of how these strategies have been employed in teaching are provided; in several cases, this has entailed development of new experiential learning and scaffolding technologies that have been made freely available online. These strategies and tools have been integrated into a new curriculum of lectures and activities for teaching supervised learning [63] to students in creative disciplines. This curriculum has been used in some of the first courses in the world focused on ML for creative practice: a MOOC, and university courses for undergraduate and masters students studying creative computing and digital arts (described in Section 5, with curriculum details in the Appendices).

Section 6 provides some initial analysis of three student activities in these courses in order to better understand the student learning and engagement supported by this curriculum. This analysis contributes to understanding the efficacy of some of the representations, explanations, and methods of informal assessment used in the curriculum, and it reveals information about student conceptions and mistakes. It also investigates how some students thought about and used ML in their creative work at the end of the course. Analysis of each activity is accompanied by reflections on its implications for future teaching and research. Section 7 draws on the previous sections and prior work to identify future research needs to further understand creative ML education, and to argue for an expanded list of pedagogical content knowledge [66] components for ML teaching.

2 BACKGROUND

2.1 ML in Creative Work

Throughout this article, I use the terms "creative practitioner" and "artist" to refer to people creating ideas or artifacts in a broad set of domains. They include creators in the fine arts, music composition and performance, and theater and performance art, as well as creators of new indie games and "makers" of other hard-to-pigeonhole artifacts and experiences. The teaching described here is aimed at students spanning all these domains, who may have various levels of expertise, and who engage in creative activities for personal enjoyment as well as for professional recognition or profit. The remainder of this section provides an overview of how and why ML may be used in these types of creative practices.

2.1.1 Creative projects may use ML for conventional or unconventional reasons. In many creative applications, ML is used for the same reasons it is conventionally used in most other domains, for instance to build models from data in order to accurately apply predictions to new data. Creative domains involve many types of data (e.g., audio, image, human movement) that are difficult to analyze computationally without ML, so ML can enable creative practitioners to work with this data more accurately and efficiently. For instance, ML might underpin a computer vision system that identifies the number of faces looking at a digital art piece, enabling the piece to change its behavior in response to the number of people present. Or ML might be used to build a computer listening system that identifies the musical phrase a musician is performing, triggering the automated playback of accompanying material [61]. There is also a long history of creative works whose content is influenced by data from "non-creative" domains[1], such as works that use data from the natural world [18] or social media [16] to drive changes in sound. ML offers powerful tools for reasoning about such data, in creative and non-creative domains alike.

Other creative work employs ML for reasons that are quite different from the those above. For instance, deep learning algorithms capable of generating new visual art, music, and other content have received much media and research attention in the past few years, and creative content generation has also been performed using simpler methods like Markov processes for decades [2, 46]. An artist using generative algorithms may rely on the accurate modeling ability of the algorithm to produce new content that is "similar" to the training set, but this is embedded in a context of broader artistic goals, such as generating content that is striking [67], humorous [65], or likely to elicit creative behaviors in response from a human partner [54]. (This article does not focus on deep learning algorithms for a number of practical reasons[2]).

Supervised learning algorithms can also be useful to design completely new responsive or interactive digital systems for a variety of creative applications. The designer of a new system can specify its behavior not by programming, but by providing a supervised learning algorithm with examples of inputs to the system along with the outputs or actions he would like the computer to produce in response to those inputs. For instance, a digital musical instrument designer might provide data demonstrating how certain human movements, perceived via sensors, should result in changes to the control parameters of a sound synthesis algorithm [47]. An audiovisual performance artist might provide examples demonstrating how different live sounds should result in changes to a generative graphics program. Or the creator of a new dance piece could provide examples of dancer movements that will trigger changes in stage lighting. For such applications, there may not be a pre-existing "ground truth" dataset; the creation of a dataset that accurately communicates the designer's intentions for the system becomes a crucial part of the machine learning process.

2.1.2 Interactive ML can support creative work. When a creator is capable of generating reasonable training examples (e.g., when the creator can anticipate and replicate the variations likely to be contained in future data, or when building a system that will respond only to the actions of the creator herself), interactive machine learning (IML) approaches [20] can become very useful. Using IML, a creator can iteratively provide training examples, train a model on those examples, test the model on new examples to identify errors or model behaviors they would like to change, and then provide additional training examples that are likely to improve the model—for instance specifying correct behaviors for inputs handled incorrectly (Figure 1). Many machine learning tools aimed at creative practitioners support such an IML approach [10, 13, 25, 30].

By allowing iterative changes to the training data, IML approaches also make it easy for creators to experiment with different design ideas and communicate their evolving goals for what should

[1]The term "non-creative domains" is used as shorthand in this article to refer to domains in which creative expression is not a primary goal. This should not be interpreted as implying that more conventional ML application domains such as science and engineering do not involve substantial human creativity.

[2]Most significantly, at the time of teaching these courses, most deep learning tutorials and tools required extensive math knowledge (e.g., linear algebra, multivariate calculus), numerical programming proficiency, proficiency with Docker, and access to GPUs or expensive cloud computing resources. These requirements exclude many of the creative practitioners targeted by the courses described here. Yet work is already being done to create more accessible deep learning tools (e.g., ml5.js [38]) and learning resources (e.g., Google's Machine Learning Crash Course [31]).

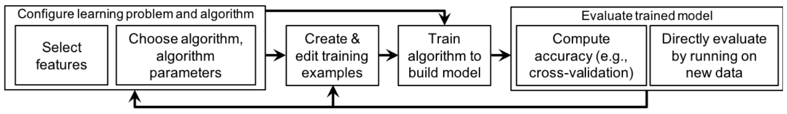

Configure learning problem and algorithm
Select features
Choose algorithm, algorithm parameters
Create & edit training examples
Train algorithm to build model
Evaluate trained model
Compute accuracy (e.g., cross-validation)
Directly evaluate by running on new data

Fig. 1. Interactive machine learning is a human-in-the-loop approach involving iterative modification of training examples, algorithms, and features.

be learned [21]. This is crucial, because in any creative domain, creators' goals are likely to evolve with experimentation [62] (e.g., as they discover that the interaction in their installation does not feel quite right now that they have built it, or as they decide they would like to know what a four-gesture classifier feels like now that they have seen what a three-gesture classifier can do).

Of course, IML is not always a suitable approach for applying ML in creative work. When the goal is to accurately model an existing dataset, or to model a phenomenon for which a creator cannot easily create or curate appropriate examples (e.g., to accurately model states or actions of diverse people), a practitioner may need to use more conventional ML approaches to improve ML performance (e.g., choosing a different algorithm or features).

2.1.3 Supervised learning offers several benefits to creative work. ML's ability to accurately model complex relationships in example data may be useful even when ML is used for unconventional purposes such as designing new interactive systems "by example." ML can make it easier to accurately model data that is noisy and/or high-dimensional, yielding model functions that better represent creators' intentions than functions implemented in programming code. Instantiating new systems from examples can bring system designers other potential benefits, as well, including facilitating faster implementation and iterative modification compared to writing and editing code, making it possible for non-programmers to build sophisticated systems, and making it easier to build systems that understand or respond to embodied practices such as music or dance (which are often hard to describe in code but easy for practitioners to demonstrate) [21].

2.2 STEAM and Creative Computing Education

Interest in teaching computing skills using creative applications of technology is growing as a result of the "STEM to STEAM" movement, which advocates for the inclusion of the arts and design alongside the STEM (science, technology, engineering, and mathematics) disciplines that are more often the focus of research and policy initiatives. As Yee-King et al. [73] note, creative arts education often engages students in "inductive, exploratory process[es] driven by self-defined goals," and the perspectives of experiential education [17], constructionism [33], and inquiry-based learning [19] all suggest that engaging STEM students in such processes is likely to be beneficial. Some STEAM teaching also aims to exploit the capacity for creative applications to motivate students from diverse backgrounds to engage with technology [27, 59].

Pairing the teaching of programming and other CS skills with creative instruction is also a key feature of programs in "creative computing" and "creative coding." Such programs typically target students aspiring to use computers to make new creative work—for instance as digital artists, electronic musicians, or creators of experimental theater or games. A number of universities (including Goldsmiths University of London, where two of the courses described in this article were taught) now offer degrees in creative computing or similar topics. Many online courses have also emerged to serve such students: for instance, MOOC provider Kadenze currently offers over two dozen creative computing courses [37]. (Note that "creative computing" involves people using computing in creative activities, whereas "computational creativity" is the term for using computational approaches to mimic human creative actions [36].)

	Paper section					
PCK Component	2	3	4	6.1	6.2	6.3
G1: Why teach this subject?	X					X
G2: What should be taught?		X	X	X	X	X
K1: Useful representations for concepts in machine learning			X	X		
K2: Effective analogies, examples, and explanations of machine learning			X	X		
K3: Knowledge of which concepts in machine learning are difficult and why					X	X
K4: Knowledge of conceptions that learners bring to learning machine learning				X		
K5: Methods of informally assessing knowledge of machine learning concepts			X	X	X	
K6: Common mistakes the learners make when applying machine learning				X	X	X

Table 1. The first column lists components of pedagogical content knowledge from Grossman (G1, G2) [32] and Ko (K1–K6) [42]. Subsequent columns indicate the sections of this paper relating to each component.

.

Many creative computing programs aim to prepare students for innovative practices in which proficiency with mass-market tools such as PhotoShop or Logic Pro is insufficient, so their students require a number of the same skills as CS students who will become technology creators in other domains. Creative computing programs typically include a substantial focus on programming using environments such as OpenFrameworks, Processing, Max/MSP, or Unity, and they often teach physical computing with platforms like Arduino. A few organizations such as the Processing Foundation [26] support the development of software tools and educational resources for creative computing, but little research has explicitly studied the learning and teaching of computational skills within creative computing courses. Further, most research on STEAM approaches to CS teaching (e.g., [27, 73] has focused on teaching programming. This paper contributes to this space by showing how other CS topics can be adapted and evaluated within a creative computing context.

2.3 ML Education

Little published research examines how to teach ML effectively to any population. One participant in a workshop on machine learning education at the 2017 International Computing Education Research conference, Andy J. Ko, published a version of his workshop paper as a blog post [42]. In it, he writes "We still know little about what students need to know [about ML], how to teach it, and what knowledge teachers need to have to teach it successfully. To correct this, I argue that we need to discover the pedagogical content knowledge (PCK) necessary for teaching concepts in machine learning." According to Shulman [66], PCK "embodies the aspects of content most germane to its teachability," including "the most useful forms of representation of those ideas, the most powerful analogies, illustrations, examples, explanations, and demonstrations," and "an understanding of what makes the learning of specific concepts easy or difficult" [p. 9]. The concept of PCK has been used in CS to understand and advance the teaching of a variety of topics—including programming, algorithms, and problem-solving—at secondary and tertiary levels [35]. However, the concept does not appear to have been employed in published research on ML teaching.

Ko [42] lists six desired components of PCK for ML, shown in Table 1 as K1–K6. Each of these components is relevant to teaching ML to creative practitioners. Yet, as Hubbard [35] notes, different researchers have conceptualized PCK differently; there is no definitive enumeration of what PCK entails, even within CS education. Other formulations of PCK suggest additional components that seem relevant to understanding creative ML teaching. In particular, Grossman's [32] reformulation of PCK includes knowing why a subject should be taught and what should be taught. These seem important to understand when considering the teaching of ML to students whose goals and backgrounds differ from the computer scientists for whom ML classes are typically designed.

This article aims to contribute to developing an understanding of PCK for Creative ML, and of PCK for ML more broadly. Section 2 has previously argued for *why* it is useful to teach ML to creative practitioners. Section 3 will next outline a proposal for *what* students should learn, and Section 4 will propose strategies for *how* this content may be taught, including specific examples of representations, explanations, and informal assessments. Sections 6.1–6.3 present brief analyses of three student activities: an early brainstorming exercise by MOOC students, a mid-course lab activity by undergraduates, and final creative projects by masters students. The analyses provide some initial evidence for the efficacy of the approach to teaching proposed in Sections 3–4; they also also help deepen understanding of how students thought about and used ML, of ways the abstractions and activities influenced and revealed student learning, and of learners' conceptions and mistakes. The analyses thus inform future pedagogical improvements and a research agenda for ML education in creative domains and beyond. Table 1 summarizes these relationships between PCK components from Ko and Grossman and the sections of this paper.

3 PROPOSED LEARNING OBJECTIVES FOR CREATIVE ML

Numerous frameworks have been proposed to characterize the processes people employ when engaged in the creative design of a new artifact. Howard, Culley, and Dekoninck [34] review a large body of literature in engineering design as well as cognitive psychology research about creativity, and they synthesize a single description of a "creative design process" that characterizes innovative artifact creation across a variety of domains. This process consists of five main "design operations": *formulation* (i.e., specifying the desired behavior of the system, based on pre-established design requirements), *synthesis* (implementing a system that aims to achieve this behavior), *analysis* (observing the implemented system's actual behavior), *evaluation* (comparing the observed behavior with the desired behavior), and *documentation* (producing the artifacts necessary for manufacturing the final product). The process model emphasizes that creators also *reformulate*—iteratively changing the implementation, formulation, or design requirements, then cycling back through the other operations in a non-linear way. (Notably, this model is quite similar to the "design thinking" frameworks commonly used to describe creative problem-solving; for instance, the Stanford "d.school" framework entails Empathize, Define, Ideate, Prototype, and Test activities [8]).

Learning objectives for any student creating new systems with ML (whether or not they are working in a "creative" domain such as the arts) can thus be derived by considering what ML knowledge or expertise is necessary to most effectively engage in each of the above operations, as well as the necessary preceding work to establish the design requirements. These objectives can then be refined for students in a particular domain of practice. In creative domains, the use of ML in a completed work seems unlikely to appreciably change an artist's approach to performing, exhibiting, or presenting the work, so objectives related to *documentation* may not be needed. However, an awareness of how ML has been used in other creative work is useful in enabling artists to contextualize their work (and is also likely to be practically helpful in demonstrating the variety of ways creative ML projects can be conceptualized and implemented), so an additional learning objective entailing awareness of other work seems appropriate. Table 2 thus enumerates the set of learning objectives established for creative ML students studying supervised learning, of which objectives LO1–LO7 are also applicable to other students building ML systems in "non-creative" domains.

Design Operation	**Learning Objectives**
Establishing design requirements, Formulation	*LO1.* Understand the structure of supervised learning problems and the capabilities of supervised learning algorithms (e.g., recognize that models compute outputs from inputs, and that algorithms create models from examples of input/output pairs; recognize that classification models output discrete labels). *LO2.* Identify feasible uses for ML in new projects, and map a new idea for applying ML onto the structure of supervised learning (e.g., map real-world problem characteristics onto inputs, outputs, training data, model).
Synthesis	*LO3.* Reason about properties of learning algorithms, data, and the problem domain to make sensible choices about learning algorithms and features for a new project. *LO4.* Apply knowledge of ML workflows and practical skill with an existing ML tool/library to create a ML model by generating/curating training data, then employing an algorithm to build a model from that data. *LO5.* Use appropriate mechanisms to pass data between the ML tool/library and the other components of a project (e.g., sensors, software for sound or art, filesystem, etc.), in order to integrate ML into fully-functioning projects.
Analysis, Evaluation	*LO6.* Choose appropriate methods to evaluate a trained model against the design criteria most relevant to the project, and apply these within the ML tool/library.
Reformulation	*LO7.* When a model implementation does not satisfy these criteria, reason about and exercise appropriate mechanisms to improve it (e.g., by changing the training data, features, or learning algorithm, within the ML tool/library).
Other	*LO8.* Understand ways ML has been used in other creative work, and draw on this to contextualise one's own work.

Table 2. Proposed learning objectives for creative students studying supervised learning, organized by the "design operations" in [34] that they support.

4 STRATEGIES AND RESOURCES FOR SUPPORTING STUDENT ACHIEVEMENT OF LEARNING OBJECTIVES

Due to the lack of research on creative ML education, there is no existing roadmap for how a course might support students in achieving these learning objectives. This section therefore draws on prior research in teaching and learning, known practices in ML teaching, and knowledge of creative ML practices to propose a set of strategies for supporting learning. It also describes how these strategies have been integrated into the design of new lectures, activities, and technologies for creative ML students. Section 5 summarizes how these components have been employed a MOOC (which was developed first) and two university courses (which were taught using materials adapted from the MOOC). Syllabi for these courses appear in Appendices C and D. In presenting these resources in detail, this article aims to contribute to a "case literature" on ML teaching, following Shulman's advocacy for educational cases to "provide teachers with a rich body of prototypes, precedents, and parables from which to reason" [66, p. 14].

The strategies described below aim primarily to support creative computing students and creative professionals who wish to use ML to create work like that described in Section 2.1. Little other student expertise is assumed; most notably, these strategies have been developed to ideally support students who may not have extensive prior programming or math knowledge. While students without programming expertise will undoubtedly face barriers to making highly original creative projects, it seems possible to support their achievement of the learning objectives through the use of GUI-based tools and pre-compiled examples for connecting ML to interesting data sources (e.g., sensors) and simple multimedia programs (e.g., for influencing sound, visuals, or games).

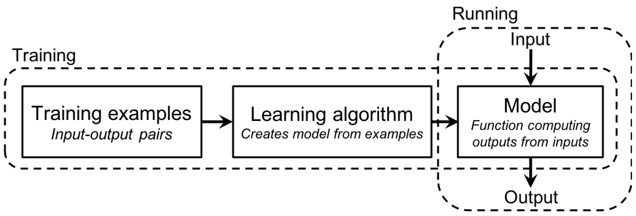


Fig. 2. The ML Pipeline abstraction used extensively in teaching.

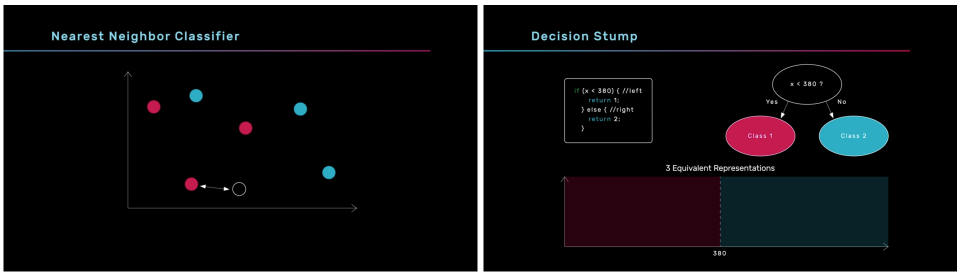
4.1 Teach Appropriate Abstractions

Ben-Ari [3, p. 259] writes of a significant obstacle to learning and constructivist engagement with programming for beginning CS students: the fact that students do not "come to class with an effective model of a computer," thinking instead of computers as "grossly anthropomorphic 'giant brain[s]'." Pea [57, p. 32–33] describes "the idea that there is a *hidden mind* somewhere in the programming language that has intelligent, interpretive powers" as a "superbug" that drives misunderstandings in beginning coders. These obstacles can be combatted by teaching students the abstraction of a "notional machine," which Sorva describes as "an idealized abstraction of computer hardware and other aspects of the runtime environment of programs" that "serves the purpose of understanding what happens during program execution" [69, p. 3]. It is plausible that beginning ML students likewise may not begin with an effective model of what supervised learning is, and they may be especially susceptible to the fiction that ML has a "hidden mind." Therefore, early teaching of appropriate abstractions seems important to help students use ML effectively.

In order to work effectively with existing supervised learning tools and libraries, and in order to reason about the structure of problems in which supervised learning may be applied, students should arguably be familiar with—at minimum—the concepts of a *model* (e.g., as a function that computes output values from input values), of *training examples* (pairs of input and output values), and of a *learning algorithm* (a thing that creates a model function, using the training examples to inform what sort of function it should make). Further, these concepts can be understood as having a strict sequential relationship: an algorithm must create a model via the "training process" before the model can be used, and after training has occurred, the model does not change.

Figure 2 shows a visual representation of these concepts and the relationships between them. In the courses described here, this collective abstraction is referred to as the "ML pipeline." This abstraction has been used in teaching elsewhere (e.g., [64]). The "pipeline" terminology is also used in some ML programming environments [49] and in writing about ML tools [55], where it may alternatively emphasize the whole sequence of data processing operations that transform raw data into the feature representation used by a model, followed by applying the model (and potentially also post-processing model outputs). Here, though, the abstraction is kept deliberately simple so that it can be introduced at the beginning of a course; it may be made more complex later on when topics such as feature engineering are introduced.

This abstraction is too simple to accommodate all supervised learning paradigms (e.g., online learning and active learning do not fit here). However, it can describe basic classification and regression, as well as sequence labeling and segmentation as performed by hidden Markov models or dynamic time warping [53]. With this abstraction, students do not need to know anything about how a learning algorithm actually builds a model in order to understand which sequences of actions with ML are sensible, or to reason about the training data and learning algorithm as two separate components (either of which might be changed to cause a change to the model). And this very



```
if (x < 380) { //left
   return 1;
 } else { //right
   return 2;
 }
```

Fig. 3. Stills from the animations and video used to illustrate nearest-neighbor (left) and decision stumps (right).

restrictive formulation does not leave much room for egregious anthropomorphizing: for instance, it doesn't easily accommodate a machine that imagines new artwork.

In the creative ML classes discussed here, the ML pipeline was introduced as the first major topic in the first lecture. Section 6.1 explores how this abstraction enabled new MOOC students to reason about what types of systems they could build with supervised learning (learning objectives LO1, LO2), without yet learning about how algorithms build models from data.

4.2 Explain Learning Algorithms Without (Much) Math

A deep understanding of ML theory or algorithm implementation does not directly support any of the learning objectives in Table 2. However, some understanding of how algorithms work can help students reason about which algorithms are most suitable for a particular problem, or about how to adjust algorithm parameters (e.g, k in k-nearest neighbor, or the complexity constant of a support vector machine) (LO3). Furthermore, a thorough understanding of how some very simple algorithms build models from data may be helpful in preventing students from attributing a "hidden mind" to ML algorithms. For instance, it may be preferable for students to reason about most ML algorithms as "better" ways of doing what they know a nearest-neighbor or decision stump does, rather than as magical black boxes that might be doing anything at all inside.

For students comfortable with algebra, geometry, probability, and calculus, these mathematical tools can be useful for clearly communicating how an algorithm works and when it may or may not be well-suited for a particular problem. And for students who are good programmers, writing an implementation of a learning algorithm may also deepen their understanding of how it works. However, it seems reasonable to hope that students without these skills could also develop enough intuition to make sensible decisions about which algorithms to use, especially if they are also able to experimentally compare algorithms when they are unsure. (After all, even professional software developers with high ML proficiency rely heavily on experimentation to guide decisions about algorithms and features to use in their software [56, Ch. 3]).

The video lectures for these courses therefore employ the following approach to describing classification algorithms to students with a practical interest in ML and varied math and programming expertise: (1) Nearest-neighbor and decision stumps are first explained in detail using animations of their operation in 2D feature spaces (Figure 3). The expectation that students can perfectly understand these algorithms is communicated and tested by intermittently asking students to take on the role of the algorithm: i.e., finding the nearest neighbor point, or deciding where to draw a horizontal or vertical decision boundary. Decision stumps are also explained as if-then-else

SVM Scorecard
+ Very powerful: can work well on complicated problems
+ Several kernel types; one might work even if another doesn't
+ Computation on new examples is fast
– Training involves solving optimization, can be slow (especially for large datasets)
– Choosing kernel and its parameters can be frustrating *(In practice, may try many combinations and choose the best)*
– Might not work well with small training sets, or with very different numbers of examples in each class.

kNN Scorecard
+ No training time
+ Capable of arbitrarily complex boundaries
+ Changing k adjusts smoothness
– Running in real-time can be slow if you have many training examples *(but most implementations use tricks to speed this up)*
– Can be hurt by noise
– Can be hurt by irrelevant and redundant features
– Problematic with very high number of features, because all examples are very far from each other

Fig. 4. "Scorecards" summarize practical considerations for each algorithm.

statements to help programmers understand that model functions act essentially like the functions they write in code. (2) Other classification algorithms are then described at a higher level, also relying heavily on visual representations of the learning process and the resulting decision boundaries in a 2D feature space (Figure 3; Appendix A). For instance, AdaBoost [28] is described in an animation in which a sequence of simple decision stump boundaries are fitted to examples whose importance (represented visually by size) is adjusted over time according to the accuracy of previous iterations. The basic principle of support vector machines [11] is described by drawing a training dataset on a deflated balloon, then stretching the balloon into a third dimension so that the two classes of examples can be separated with a plane (here, a pair of scissors). (3) Equations are used sparingly and always presented as a way to deepen understanding for students who are comfortable with the notation. For instance, the equation for Euclidean distance is presented for nearest-neighbor, to illustrate that moving from a 2D feature space to a higher-dimensional one is a simple matter of adjusting the distance equation. (4) The description of each algorithm is concluded with a "scorecard" (Figure 4). Rather than summarizing the learning procedure or theoretical motivation of an algorithm, the scorecard summarizes the algorithm's practical benefits (and drawbacks) as they may intersect with properties of the learning problem or user goals.

The explanation of regression is similar. Linear regression is first explained in detail. Students are asked to imagine how they would fit a line to a visualization of points in two dimensions; this drives a discussion of what it means to "fit" a line, and when a straight line may prove too simple a model. Polynomial regression and neural networks are then presented as increasingly flexible methods for fitting points with lines of other shapes. Optional supplemental lecture videos are available for interested students; these motivate the problem of finding a line of best fit as an optimization problem and describe at a high level how gradient descent is used to solve this optimization problem for neural networks. Likewise, the explanation of sequence modeling algorithms relies heavily on visual representations of the algorithms working on low-dimensional data (Appendix A).

This visual understanding of how algorithms create models in low-dimensional feature spaces is reinforced and explored through hands-on activities described in Section 4.6. Visualizations of algorithms in higher-dimensional feature spaces are not attempted. Instead, students are exposed to higher-dimensional problems through lecture demonstrations and hands-on activities using models that work with larger numbers of features from sensors, webcams, and audio.

4.3 Recognize Distinct Knowledge Needed by Creative Practitioners

Most learning objectives in Table 2 are applicable to people building ML systems in any domain. However, using ML in creative practice requires a distinct (and arguably broader) set of knowledge than using ML in other domains. First, many creative practitioners may benefit from using IML approaches (Section 2.1.2) in which they iteratively change the training data to improve and refine models. Students should therefore be taught about IML and about how to reason about whether it is appropriate for a particular context. There are also human skills that can lead to IML being more effective, such as creating training data as free from noise as possible, or (when feasible) choosing phenomena to classify that are easy to discriminate within the given feature space.

Second, prior research has shown that cross-validation can be a surprisingly poor measure of model quality within certain IML contexts, where training sets may be very small and creators may deliberately place examples to have particular effects (e.g., placing examples close to intended decision boundaries) [24]. IML users may sometimes benefit more from creating new test data on the fly and observing model behavior on that data, as this allows them to assess models against diverse and subjective criteria (e.g., presence of mistakes in high- or low-stakes areas of the feature space, or the comfort or "interestingness" of a new gestural controller). Further, the emphasis on generalization accuracy as the top priority, and on using metrics such as cross-validation that estimate generalization accuracy, is inappropriate for some (though certainly not all) creative ML applications [24]. Creative practitioners should therefore learn about a variety of model evaluation strategies and learn to reason about which strategies are appropriate for a given context.

Third, many creative practitioners are interested in working with data from sensors, audio, images, and video. These data present particular challenges, in that raw data (e.g., audio samples, video pixels) are often not a suitable representation to pass directly to ML algorithms. (An exception arises with certain deep learning approaches that can learn good representations directly from raw media data—e.g., [44, 71]—but this is infeasible for many creative applications in which no large datasets exist.) Therefore, creative practitioners require instruction on basic strategies for "feature engineering"—i.e., applying common transformations to the raw data to render it possible to learn from. This can include simple operations such as segmentation, thresholding, or computing average or standard deviation over a sliding time window; processing with digital filters; or applying transformations commonly used to extract information that is relevant to human perceptual processes. These include the computation of Fast Fourier Transform (FFT) or Mel-Frequency Cepstral Coefficients (MFCCs) [48] from audio, or edge detection or optical flow from video.

In the courses described here, IML was the primary approach discussed in lectures and used in lab activities. Lectures described contexts in which IML is and is not appropriate. Although the concept of generalization accuracy and the use of cross-validation were presented in detail, lecture content also discussed circumstances in which cross-validation accuracy may be an inappropriate metric. Hands-on lab activities in which students experimented with IML (e.g., Figure 5) were the primary mechanism for learning how to provide "good" training data and to reason about what is easily learnable with given algorithms and features. Some lectures also provided practical tips on how to structure ML problems to be easily learnable (e.g., describing how multiple simple classifiers could be used in parallel instead of a single complex classifier, for labeling tasks entailing combinations of non-mutually-exclusive labels).

One of the seven video lectures was exclusively dedicated to feature engineering for sensors, sound, and video. Furthermore, a number of scaffolding tools were developed to enable students without programming or signal processing expertise to meaningfully experiment with multimedia features and feature engineering. A GUI-based software tool called Input Helper[3] was developed, which allows students to interactively apply common operations to raw data (e.g., average, standard deviation, first- and second-order difference, digital filter equations, and mathematical operations entered as text formulas). A number of standalone, real-time extractors for common audio and video features (e.g., FFT, MFCCs, facial keypoints) were developed and provided to students as executables and source code in OpenFrameworks and Max/MSP.

[3]http://www.wekinator.org/input-helper/

A4.1. Use the Classification Explorer to attempt to recreate the two sets of decision boundaries at right, using each of the following classifiers: kNN, AdaBoost, decision trees, SVM. Change the training data and/or algorithm parameters to try to match the decision boundary as well as you can. Take a note of the following, for each algorithm: How sensitive is the algorithm to outliers and small changes in the training data? How does the model improve (or not) as you add more examples or change algorithm parameters? How well were you able to match the target decision boundary? Make a note of how difficult (on a scale of 1 to 5) it was to create this decision boundary with this algorithm.
A4.2. Build a classification system you find useful and/or interesting, making use of example feature extractors or output programs from the Wekinator website if you wish. Spend time experimenting, spending at least 30 minutes and/or trying at least 5 classifiers. Once you're happy with your classifier, write your answers to some questions, including: Describe in a few sentences how you intend your classification system to be used, and why you find it useful and/or interesting. What was the biggest challenge you encountered in making the system work the way you wanted it to? And how did you try to overcome this challenge? If you were to give advice to another student trying to build a similar system, what advice would you give?
A4.3. Post your answers to the previous questions, or another question or comment, to the online forum.

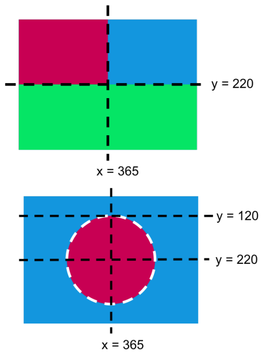


Fig. 5. Abbreviated instructions for MOOC assignments A4.1–A4.3 (see Appendix C)

4.4 Use Technologies Appropriate To Creators

Achieving the learning objectives entails hands-on proficiency applying ML in one's own creative work. Enabling students to develop this proficiency is challenging when students work in diverse domains with different digital tools and have varying levels of programming expertise. Effort was made to ensure that the technologies chosen for illustrating concepts in teaching, as well as supporting hands-on activities and student project work, could support both a wide variety of creative practices and appropriate approaches to using ML.

These considerations underpinned the choice of the Wekinator software [25] as the primary ML tool to support lectures, activities, and projects in these courses. Wekinator provides general-purpose learning algorithms for classification (kNN, AdaBoost, SVM, Decision Trees, Decision Stumps, Naive Bayes) and regression (linear and polynomial regression, neural networks), and dynamic time warping (DTW) for sequence matching. It encourages an IML workflow, enabling users to create new training examples in real-time, and to iteratively re-train on modified training sets. It is GUI-based and accessible to non-programmers, but it still provides fine-grained control over ML (e.g., setting algorithm parameters and selecting features can be done in the GUI). It allows the creation of complex creative systems with ML. For instance, multiple models can be created within a single project, and they can be trained individually or as a set; multiple Wekinator projects can be run in parallel or in series; trained models from one project can be loaded into another.

Wekinator uses Open Sound Control (OSC) [72] to connect to a wide variety of sensors and software environments. It uses OSC both to receive feature values and send model outputs. (That is to say, the two vertical arrows in Figure 2 connect the feature extractor to Wekinator, and Wekinator to the output process, via OSC messages.) OSC is supported in most creative computing environments and conventional programming languages (e.g., Processing, OpenFrameworks, Max/MSP, PD, Unity, Python, C++). Many creative programmers already use OSC. The Wekinator website [23] contains dozens of examples of compatible feature extractors and output processes in many programming languages; these may be run as-is or serve as example code for programmers to modify. New examples for students in these classes were added to this site as well.

Alternative toolchains could support different cohorts in achieving the same learning objectives, while presenting different tradeoffs. For instance, ml.lib [10], RAPID-MIX API [4], and scikit-learn [58] can be used by students who are proficient programmers to employ many of the same learning algorithms, without the need to run multiple programs communicating via OSC. Yet to support the same type of lab activities and creative projects, students or instructors would need to write significant additional code to implement functionalities such as control over iterative IML data collection, training and evaluation; interfacing with sensors and other data sources; and connecting to other creative software (e.g., game engines or music environments).

4.5 Integrate Creative Perspectives

Seeing examples of creative work using ML should help students better understand how ML can be used creatively (LO2) and gain familiarity with the landscape of existing practice (LO8). Several lecture videos therefore include demonstrations of ML systems used in real-world performances, with technical descriptions of how they use ML as well as discussion of how the implementation supports the works' creative aims. Further, approximately 1 hour of the last video lecture is led by composer Laetitia Sonami. In this segment, titled "Developing a Practice with Machine Learning," Sonami discusses why she uses ML in her practice and describes how she implements several of her pieces using Wekinator and Max/MSP.

We can also view students' creative experiences and practices as resources that may be drawn on to deepen their knowledge of ML, similarly to how Moll et al. propose "ethnographically informed classroom practices" that draw on the "funds of knowledge" within students' households and communities [51]. One way the creative ML courses described here attempt to do this is by telling stories that map abstract considerations about ML into a space that is both physical concrete and creatively relevant: the stage. The 2D feature space visualization used to ground discussion of classification algorithms, features, and noise (Section 4.2, Figure 3) is initially described as a stage, and each data point represents a dancer standing in a specific position on that stage. In this environment, a classifier could be used to change the sound or stage lighting according to where the dancer stands, and its decision boundaries partition the stage into segments. Different classification algorithms present different trade-offs regarding the possible shape of the decision boundaries, the number of examples needed to build a useful classifier, and the quality of the classifier that is possible when the sensor acquiring dancer position is noisy.

4.6 Support Experiential Learning of ML

Most of the learning objectives in Table 2 seem ripe to be addressed via activities that support experiential learning [43], in which learners iterate through a cycle of concrete experience, reflective observation, abstract conceptualization, and active experimentation. A number of technologies were therefore created to support experiential learning with ML regardless of math or programming expertise, and activities were structured around these technologies to explicitly support cycles of experimentation and reflection. For instance, a "Classification Explorer" software application (Figure 6) was developed to allow students to interactively create training examples in a 2D feature space (analogous to the "dancer's stage" described in lecture), train a classifier on this dataset, plot the decision boundary of the classifier on the same space, then explore how the classification boundary changes in response to changes in the training data, algorithm, or algorithm parameters. A "Regression Explorer" was similarly created to enable students to experiment with regression in a 1D feature space (Appendix B), and a "DTW Explorer" was created for experimenting with sequence labeling of shapes drawn with the mouse. The Input Helper (Section 4.3) was developed to enable engineering of higher-level features from raw data. A number of new example feature extractors were released as executables to enable students to experiment with different sensors and input modalities. Similarly, executables were released that use outputs from Wekinator to drive changes in graphics, animations, music, sound, and games. Students who are not strong programmers can thus still experiment and build many novel systems with ML. Students who are programmers can efficiently experiment without worrying about code, and can also adapt the source code from these many examples for their own use. All technologies developed for these courses have been released as open-source software and are available on the Wekinator website.

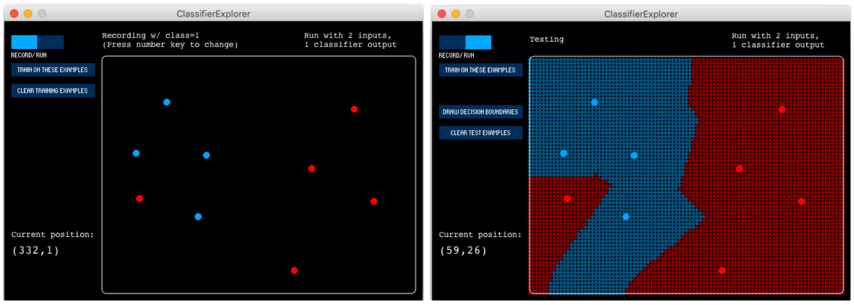


Fig. 6. The Classification Explorer interface allows students to record new training data by clicking on screen (left) and to evaluate the trained model by either clicking to add test points individually, or automatically generating many test points to illustrate the decision boundary (shown at right). Here, a 1-nearest-neighbor classifier is used.

Several hands-on activities were designed to intersperse use of the "Explorer" programs with reflective exercises (often with a social component involving peer discussion) and building small creative projects with sensors or other real-time data. For instance, Figure 5 shows an abbreviated version of the instructions for the fourth MOOC assignment (also employed in the university course's labs), which took place after all classification algorithms had been introduced in lectures.

An automatic assessment system ("autograder") was also developed for all MOOC lab activities, and made available to University students as well, to provide students with feedback to supplement their own reflections. For instance, the autograder used Wekinator logs to compute an accuracy score for each classifier created in the MOOC assignment in Figure 5 (computed by applying the logged classifier to unseen examples matching the target boundaries). Students had the option of submitting activities to the autograder multiple times to try to improve their classifiers' accuracies.

5 USING THESE STRATEGIES AND MATERIALS TO TEACH THREE NEW CLASSES

5.1 MOOC

A MOOC titled "Machine Learning for Musicians and Artists"' [22] was the first course developed using these strategies. Appendix C details the organization of video lectures and assignments in this seven-session course with approximately 8.5 hours of lecture content. The MOOC launched in 2016, and it is still running. As far as can be determined, this was the first course explicitly devoted to ML for creative practitioners ever to be offered. As of June 2018, 12,834 students had enrolled, 852 of whom were "subscribing" students able to submit assignments for marking (though completing the MOOC does not lead to credit at any degree-granting institution). Of these, 363 students had submitted at least one assignment and 59 had submitted all assignments. This completion rate is comparable with other MOOCs [40].

5.2 Undergraduate Class

MOOC content was used support a 10-week class titled "Data and Machine Learning for Creative Practice" at Goldsmiths, offered in Autumn 2017 as an optional elective to final-year undergraduates. Thirty-five students enrolled from the following degree programs: Creative Computing (11 students), Music Computing (11), Computer Science (7), Digital Arts Computing (5), Games Programming (1). All students had at least two years of university-level programming instruction. Math skills among these cohorts are generally not advanced (e.g., students have been exposed to summation but many struggle with it; none have taken a university course in linear algebra or calculus).

The course used a “flipped-classroom” [6] format: students watched MOOC lectures independently before coming to class, and class time was devoted to hands-on activities and discussion. Appendix D shows how the MOOC lectures and assignments were integrated into the course. Many of the MOOC assignments were used as in-class activities done in pairs. The autograders used for the MOOC were integrated into Goldsmiths’ online learning environment so that students could submit MOOC assignments for automated formative assessment. A number of additional activities were added to the undergraduate course. For instance, in one set of activities, students designed their own “decision trees” for choosing an ML algorithm; this is discussed further in Section 6.2.

All students independently completed a final creative project. The project brief was open-ended, accommodating creative ML projects in any domain and built with any tools. Supplemental resources (including web tutorials and alternative in-class activities) were created for the seven students who expressed a strong interest in content generation with deep learning, and who self-identified as having the technical skills necessary to install and program with TensorFlow [1].

5.3 Masters Class

A very similar class was run as an option for masters students at Goldsmiths in Spring 2018. Twenty-eight students chose to enroll, 22 of whom were enrolled on the MA/MFA in Computational Arts program, 3 on the “Independent Games and Playable Experience Design” MA, 2 on the MSci in Creative Computing, and 1 on the MSci in Data Science. Programming ability varied widely; the majority of students did not have STEM undergraduate degrees and many had only begun to program in the previous term. However, these students generally had well-developed creative practices, often in a professional capacity. The weekly activities and final project requirements were nearly identical to the undergraduate class. However, three of the ten in-person sessions were canceled due to a nation-wide academic strike. Students were advised to complete the in-class activities for these weeks on their own or in groups outside of the university.

6 UNDERSTANDING LEARNING AND TEACHING

In this section, student work in these three classes is examined to better understand teaching and learning in terms of the PCK components in Table 1. Section 6.1 examines a brainstorming exercise performed by 30 MOOC students at the beginning of the course. Section 6.2 examines an undergraduate activity (from 10 students) performed in the middle of the course. Section 6.3 examines 21 masters students’ final creative projects. Each of these sections concludes with remarks about threats to validity and implications for future teaching and research. Quotes are attributed to students using a “U” prefix for undergraduates and “M” for masters (e.g., “U3” is undergraduate 3).

6.1 How Do Beginners Reason About Using ML After an Introduction to the ML Pipeline and Basic Tools?

6.1.1 Motivation, research questions, and method. The first video lecture introduces the ML pipeline (Section 4.1), a foundational abstraction used throughout these courses. It also uses this concept to describe the Wekinator software (i.e., as the tool for building the models, then running the models to compute outputs from inputs), and to explain three short demos of building and running models in Wekinator. In the MOOC, this is followed by assignment A1.1, in which students train and run Wekinator using an example “input” program (the position of their face tracked with a webcam, or the position of an object on screen) and an example “output” program (in which Wekinator’s outputs influenced sound synthesis or animation). Then, in assignment A1.2, MOOC students brainstorm three scenarios in which supervised learning could be used to make a piece of interactive art or music, then write one-paragraph descriptions of each scenario. Students must post these descriptions on the assignment forum to receive credit.

This section examines a selection of these brainstorming scenario texts to explore the following questions: How do this abstraction and the accompanying explanations support students in identifying feasible uses for ML in new projects? Do the scenarios differ from the applications presented in the class? (i.e., How do the abstraction and explanation support LO2 in Table 2?) What does the brainstorming exercise reveal about learners' conceptions and mistakes, and what are the implications of these for the content that should be taught and for the future use of this exercise? (These questions correspond to K1, K2, K4–K6, and G2 in Table 1).

One scenario was selected from each of 30 randomly-selected students enrolled in the first MOOC offering. First, the author coded each scenario text as to whether it described a feasible application of supervised learning. Second, the author used an open coding procedure [15, p. 561] to identify properties of the scenarios that revealed unexpected or potentially problematic conceptions about ML. One prominent pattern that emerged was that a number of students' scenario ideas could feasibly be built without ML. Therefore, the author then coded all feasible scenarios according to whether ML was likely necessary, unnecessary but possibly helpful, or unnecessary and unhelpful. Third, the author used open coding to identify properties of scenarios that might present practical challenges should students attempt to implement them (as this could suggest a need to teach students the additional skills required for such projects, or to manage students' expectations). A prominent pattern that emerged is that many scenarios would be difficult to realize without substantial skill in feature engineering. The author therefore coded all feasible scenarios according to whether they would likely present a feature engineering challenge to a creative computing student without substantial signal processing expertise. Finally, a second researcher with expertise with creative ML and signal processing reviewed the choice and application of all codes, and all disagreements between the two researchers were resolved via discussion.

6.1.2 Findings. 27 of the 30 scenarios included a feasible application of supervised learning. Each of the 3 exceptions appeared to stem from a different misconception about the capabilities of supervised learning. One student assumed that supervised learning could learn by example to compute functions such as the coherence between two oscillating signals; in reality, this would be better implemented by explicitly programming the desired functions, since their mathematical form is known. Another student assumed that supervised learning could generate webpage layouts that dynamically adapt to users' preferences; this seems better suited to formulation as a reinforcement learning problem. A third assumed that supervised learning could be used to implement a full natural language communication system, capable of interpreting voice commands such as "move a little more to the right." A fourth student identified some feasible uses of supervised learning, but in another component of their scenario proposed that ML could be used to de-mix a complex musical track into its constituent parts. Like natural language processing, this is a very difficult problem whose complexity may not be apparent to students who have not studied it.

Scenarios described diverse applications that differed substantially from those presented in the class. Scenarios included using a microbial fuel cell to drive sound synthesis, a Dali-inspired interactive installation in which participants' movements influence the appearance of melting clocks, and a piece for a Catholic church in which congregants are illuminated with light patterns based on whether they have entered confession or performed penance. Music, sound, and art figured prominently in the scenarios; students also described ideas for health and accessible interface design.

Just 6 of the 27 feasible scenarios seemed likely to require ML: these seemed too complex to be implemented with manually-written code (e.g., analysis of brain wave patterns, identifying mood shifts in guitar playing). In 3 other scenarios, ML seems likely to be neither necessary nor helpful: these tasks could be accomplished with very simple code (e.g., detecting whether a hand is touching a sensor). In the remaining 18 scenarios, someone with sufficient technical skills (e.g., competent programming, understanding of sensors and/or signal processing) could likely implement a working version by writing code. However, these scenarios were complex enough that ML seemed potentially helpful, for instance making it easier to build a sufficiently accurate system, or making it faster to build the system. For instance, one scenario involved building a sensor glove to control a music system or video game using hand gestures. Such gloves have been made by professional musicians in the past without machine learning (e.g., [68]), but the most well-known commercial musical glove controller at present uses ML for gesture recognition [70].

14 of the 27 feasible scenarios would almost certainly present a feature engineering challenge; these involved tasks such as tracking objects in video, detecting musical pitches in audio, or analyzing EEG data. 3 other scenarios could possibly present a challenge (depending on interpretation).

6.1.3 Implications for teaching and research. The success with which students could identify diverse and feasible applications of supervised learning lends some preliminary support to the ML Pipeline abstraction and its use in the first lecture and activity. Analysis suggests that student misconceptions at this stage may include attributing too much power to ML and not recognizing the difficulty of working with certain types of data. However, nothing is known about these students' prior exposure to ML, and MOOC students who complete assignments may differ in their backgrounds and aims from the many who do not [41]. Future research should therefore study other students about which more is known, to more confidently understand how the ML pipeline and early course content shape student reasoning, and to identify misconceptions that other students may bring. Such work should also explore differences across students with different disciplinary backgrounds, since the pipeline's modular structure and notions of "inputs" and "outputs" are likely to be more familiar to students with prior computer science study.

It is perhaps not surprising that so many students proposed scenarios that did not require ML. Students at this stage are unlikely to have a precise idea of what ML is capable of, but they may have learned through experience about creative systems that are difficult to build using programming. The question of how to interpret students' use of ML in real projects for which ML is not necessary is explored in Section 6.3.3.

It seems reasonable that students who have not worked extensively with media or sensor data may lack good intuition about what information can be easily extracted from the raw data without ML, what information can be extracted by applying ML to the raw data, and what information is best extracted by applying ML to higher-level features computed on the raw data. For instance, the concept of pitch may seem simple to a human musician, but using the algorithms here to build a pitch detector requires first extracting features such as FFT from the raw audio. The number of scenarios presenting feature engineering challenges supports the argument (Section 4.3) that creative practitioners will benefit from feature engineering instruction, and from tools like the media-specific feature extractors produced for these courses. However, many student scenarios would still require more specialized domain knowledge and/or substantial experimentation to find good features, and it is hard to see how a general course in creative ML (rather than one specifically devoted to audio or video, for instance) could better meet the needs of these students. Future creative ML teaching would benefit from teaching students when and how they might exploit deep learning algorithms capable of learning good models directly from raw data (e.g., image pixels or audio samples) [44, 71]. However, for modeling problems where large datasets are not available for applying deep learning, manual feature engineering may continue to present challenges for creators and for teaching.

As an informal assessment, the scenario brainstorming reveals a great deal about students' proficiency using the ML Pipeline to generate new ideas, and about possible application areas of interest to students. However, reading and interpreting the rich scenario prose was painstaking. A

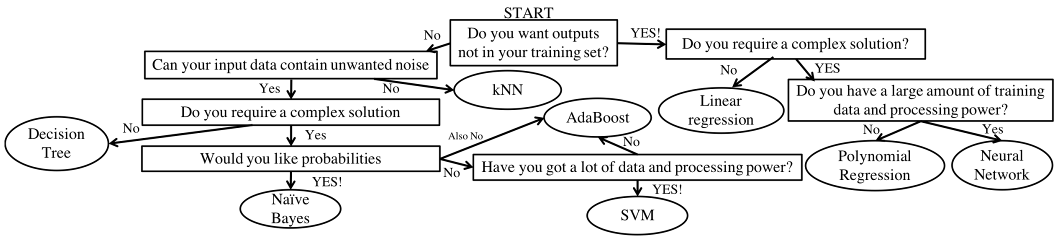

START
Do you want outputs not in your training set?
YES!
Do you require a complex solution?
No
Can your input data contain unwanted noise
Yes
No
kNN
Do you require a complex solution
No
Decision Tree
Yes
Would you like probabilities
YES!
Naïve Bayes
Also No
AdaBoost
No
No
Have you got a lot of data and processing power?
YES!
SVM
No
Linear regression
YES
Do you have a large amount of training data and processing power?
No
Polynomial Regression
Yes
Neural Network

Fig. 7. Algorithm decision tree by student U7, digitally transcribed for readability.

lighter-weight, easier-to-evaluate version of this assessment was therefore developed for future students, in which students explicitly list (1) the input data they imagine using; (2) what they imagine the model function will do; and (3) how the model outputs will be used.

6.2 What Can We Learn About Student Reasoning from "Algorithm Decision Trees"?

6.2.1 Motivation, research questions, and method. In week 3, Goldsmiths undergraduates were asked to do the following exercise independently: "Draw a decision tree (i.e., a flow-chart) for choosing a supervised learning algorithm for a new creative application. This chart should show someone how to decide which algorithm to use (of those seen so far). That is, the leaves at the bottom of the tree should correspond to one of the above algorithms (or possibly 'use something else' if none of these algorithms are appropriate)." Students collaboratively revised this decision tree in class, then updated it again in week 5 after additional algorithms were introduced. Ten students (1 female) chose to submit their final decision tree for a grade (students could choose any four in-class activities to submit). A fairly typical submission appears in Figure 7.

The ten trees were examined to answer the following questions: Do the trees present a reasonable and actionable plan for choosing an algorithm based on properties of the algorithms and learning problem (i.e., LO3 Table 2)? What do the trees reveal about student learning and misconceptions? What are the implications of these for the content that should be taught, and for the future use of this exercise? (These questions correspond to K3, K6, G2, and K5 in Table 1.)

Due to the number of nodes and leaves, it was infeasible to examine the reasonableness and actionability of each full decision path. Analysis of the trees therefore proceeded as follows: First, the author examined all 96 leaves and assessed whether the immediately preceding node led to that leaf using a decision that was reasonable, not reasonable, or possibly reasonable. (For instance, in Figure 7, it is reasonable to choose Naive Bayes if one answers "YES!" to "Would you like probabilities?") Second, the author applied an open coding scheme to all 110 decision nodes (at all levels of all trees), coding for the types of decisions employed within them. Third, the author coded each of these nodes according to whether it was actionable—that is, employing a question that could be answered using information about algorithms and the learning problem that might reasonably be available. A second researcher with ML expertise reviewed the choice and application of all codes, and all disagreements between the two researchers were resolved via discussion.

6.2.2 Findings. 66 of the 96 leaf-level decisions were coded as reasonable, 19 possibly reasonable, and 11 not reasonable. The majority of final decisions were reasonable in all but one tree.

As Table 3 shows, students drew on a wide variety of properties of the algorithms, features, data, and higher-level user goals to reason about choosing an algorithm. 9 of the 110 nodes were nonsensical or insufficiently clear to code (e.g., "Are multiple inputs combining 1 output?" [U3]).

Code description	#	Code description	#
Related explicitly to type of output data (e.g., labels vs. numeric values)	21	Explicitly mentions probability as being needed/relevant	7
Explicitly about classifier decision boundary shape	10	Explicitly about regression line shape	9
Otherwise related to complexity/capacity of algorithm	16	Related explicitly to noise or outliers	15
Related to properties of features (e.g., no. of features)	5	Explicitly related to dataset size	5
Explicitly about unbalanced training data	2	Explicitly about training or running speed	6
About some other practical requirement or preference (e.g., computing power)	13	Essentially about the algorithm strategy, rather than some property of the data or problem	25
Something else	1		

Table 3. Open codes applied to the types of decisions employed in every student decision tree node, and the count of nodes matching each one. Some nodes matched more than one code.

Otherwise, most of the properties captured in Table 3 can be helpful for informing algorithm choice in at least some contexts.

40 of the 110 decision nodes did not seem to be actionable without additional information obtained by experimentally evaluating algorithm performance. The majority of these were the 25 nodes coded in Table 3 as describing characteristics of algorithm strategy. Examples include "Would you like to classify by boundary or by nearest data point?" (U5), "Should classes be chosen by probability?" (U4), "Will it use if/else statements?" (U3). One might occasionally have an a priori reason for choosing a particular algorithmic strategy (e.g., selecting an algorithm that "uses if/else statements" so the trained model function can be interpreted by a human). But in most cases, one simply prefers the algorithm that provides the best model (e.g., has the best generalization accuracy), and it is unknown in advance what type of algorithm strategy will provide that. Instead, experimental comparison of algorithms is necessary.

6.2.3 Implications for teaching and research. Overall, this analysis suggests these students can correctly identify a variety of relevant properties of algorithms, but they may struggle to apply this knowledge to synthesize an actionable plan for choosing an algorithm in practice. This is not at all surprising; even ML experts rely on extensive experimentation rather than their ML knowledge alone to choose an algorithm for a new problem [56]. It is interesting that students did not simply insert decisions based on experimentation into their trees, and instead listed correct but unhelpful facts about algorithm strategy. It could be that they felt pressure to show that they had some relevant knowledge, even if it was insufficient to make all decisions. But in several class discussions with both undergraduate and masters students, it appeared that students were troubled that they could not deduce precisely which algorithm was the "right" one based on properties of the problem domain, and they often expressed bewilderment at statements that even experts might not know the "right" algorithm for a problem without experimentation.

The following changes are therefore planned for future teaching: (1) The importance of experimental comparison of different algorithms will be emphasized more strongly in lectures and lab exercises, so that students see this as a matter of good practice rather than a "failure" to sufficiently understand the algorithms. (2) The algorithm decision tree exercise will be revised to explicitly encourage students to choose nodes that describe decisions about the problem that they are likely to be able to answer (rather than decisions about algorithm strategy), and to encourage them to incorporate experimental comparison into their trees where necessary.

As an informal assessment activity, evaluation of the decision trees is time-consuming. A more constrained version of the activity could be created in which students choose decision nodes from a pre-defined set; this could employ paper cards to support collaborative construction of trees, or a software version could enable automatic assessment of correctness for each leaf. Such an automated version could be used to efficiently study reasoning and misconceptions by different and larger cohorts (particularly important since only 10 undergraduates were studied here), and to understand how specific lab and lecture activities lead to changes in reasoning.

6.3 How Did Masters Students Use ML in Creative Projects?

6.3.1 Motivation, research questions, and method. Masters students had to complete an independent creative final project, which could use any technologies and and any type of machine learning and/or data analysis. Students submitted a written project report and all project files (e.g., source code, Wekinator projects). Of the 26 masters students who finished the course, 21 used supervised learning in their final project (2 were female; 19 of the 21 were Computational Arts students and 2 were Indie Games students). 4 other students used generative methods and 1 did not use ML. This section examines project work by these 21 students to investigate the following questions: How and why did these students use ML? What challenges did they encounter? What are the implications of these for future teaching and research? This section thus addresses G1, G2, K3, and K6 in Table 1.

Each project report was therefore examined to identify sections that related to the following questions: What were students' motivations and criteria for success? What was the role of ML in the project? What technologies were used? What challenges did students report, and how did they navigate them? How successful did they feel they were, and what would they do if they had more time? Project files were examined to identify the algorithms and features used, and to view the history of training example creation and modification within Wekinator. The author also looked for cases in which students could have made better choices about aspects of the ML implementation (e.g., cases when choosing different features would have better satisfied their goals).

Sections 6.3.3–6.3.5 discuss some of the themes emerging from this data. First, though, Section 6.3.2 describes three projects in detail, illustrating some of the nuanced thinking and working exhibited by these students and providing exemplars to ground the subsequent discussion.

6.3.2 Three exemplar projects. **Student M9** used a Leap Motion to create a hand gesture controller for a modular synthesiser. His goal was to improve audience experience for "semi-improvisational generative ambient music": "[s]eeing a performer physically interact with whatever is generating or modulating sounds in real time would add a much more engaging dimension to any performance." At the beginning of the project, he envisioned using hand height and tilt as well as finer-grained finger movements to control sound. He had originally intended to use DTW, but after experimentation he chose regression: "I found that more interesting and usable results came from having fluctuating smooth [control signals], as opposed to the yes/no messages of DTW and the stepped numbered outputs from classification." The ML component ended up being quite simple in the final version: one neural network used all five fingertip (x,y,z) cordinates to predict hand height, and another used these same features to predict hand tilt. The main challenge he described was deciding how to translate Wekinator outputs into synthesiser controls; ultimately, he wrote his own software to smooth and scale Wekinator outputs, and to route individual Wekinator models onto control over multiple synthesizer parameters. M9 stated "Overall, I am very happy with how the project ultimately came together. It has given me a new interest in creating interesting ways to interact with synthesizers, particularly in ways that not only make the instrument more expressive, but make performing with it more engaging to the [audience]."

Student M16 used a webcam to create a "digital stress ball," in which an on-screen 3D shape distorts to reflect the level of stress detected in the user's face. Inspiration for the piece came from the "stress and frustration" he experienced in learning to code earlier that year. His intention was "to make a piece of software that was intuitive to use, where the interaction could feel physical and instinctive." He initially explored Leap Motion and Kinect to sense hand manipulations of a virtual stress ball, but he ultimately chose a webcam as he was "keen to build a system that encouraged use by anyone" without special hardware. He built three neural networks for regression, all of which used 132 facial key points (from one of the example feature extractors) as inputs. One estimated the amount of stress or distortion of the face, one the degree of tilt of the face, and one the degree of rotation (looking left or right). M16 reported providing many training examples containing subtle variations of his face, to "give a broader range of stressed faces for the algorithm to learn from." He experimented with linear regression but found the results from neural networks to be more "intuitive" with "less perceived error when using the program." The main challenge he described was the presence of jitter in the model outputs, which led to "choppy" visual behaviors; he addressed this by using the InputHelper to smooth Wekinator outputs before sending them to the animation. Ultimately, he wrote that "While it's clear the algorithm is not 100% accurate, the model has learned the basics of recognizing the user's facial gesture patterns and is able to accurately translate these into readable values for the [animation]. His stated that the project "does somewhat achieve my original design goals and creative aims," though he noted that the animation "is not as clean and smooth as could be."

In **M21**'s project, the motion of a hand-held smartphone influences an on-screen 3D particle system of flocking spheres, which can be paused at any time and sent to Blender [7] for 3D printing. A painter by training, M21 wanted to create "a tool driven by the affordances of moving your whole body through space." She desired a tool that was "playful and influenced by randomness and chance. I wanted to be surprised by the tool rather than have full control of it, creating some mappings that I could intuit rather than consciously break down into component parts while I was using it." She initially experimented with DTW but was unhappy with the level of accuracy that could be obtained; she also experimented with other inputs (including a webcam and iPad) before settling on the phone (e.g., finding that moving an iPad in the air felt "precarious"). She "found that regression made for more intuitive spatial mappings" by "mapping physical space to the simulated 3D space." Yet through experimentation, she found that "creating too 'tight' a mapping between real-world acceleration and simulated [particle] velocity made for boring and uninteresting results. It was the tying of real-world spatial motion to the flocking behavior and probability of [particle] attraction that finally pushed the tool into a playful and interesting space." She ultimately used 20 phone motion and touch features, extracted with a third-party app [45]. She crafted 4 linear regression models to each provide simple and predictable controls from one sensor dimension to one control dimension (e.g., the phone z-axis controlled the flocking algorithm sensitivity). She also used 4 polynomial regression models to implement a "many-to-many" mapping where several input features influenced a group of simulation parameters in a more complex way. M21 wrote that "I found myself spending a lot of time playing and exporting the results, which is a good sign for me... I did question whether I needed machine learning for this particular project, especially on some of the one-to-one mappings, however it did make working with the accelerometer and rotational velocity much easier."

6.3.3 How and why students used ML.

All 21 projects used ML to implement a real-time system that responded to actions or states of people or the environment. 5 students used ML to accurately model real-world categories or phenomena: this includes M16 above, as well as projects identifying common hand gestures (e.g., "thumbs-up"), modeling the relationship between a pop song's structure and popularity, and (cheekily) classifying whether an image is "art." The 16 other students used ML to model actions or concepts that they defined themselves. For instance, like M9 and M21 above, a number of projects reacted to gestures that were not part of any cultural vocabulary and that lacked intrinsic meaning. Others among these 16 used ML to map from data in one domain to another for artistic rendering; this includes turning real-time weather data from a chosen city into an abstract visualization, and turning the motions of a painter's hand into an abstract sonification of the painting process.

Two projects used RAPID-MIX [4] (a C++ library); all others used Wekinator. Students used a variety of algorithms, and several used multiple algorithms simultaneously. 14 projects ultimately used regression (4 linear regression, 3 polynomial, 11 neural network). 8 used classification (6 kNN, 3 AdaBoost, 1 decision tree). 1 used DTW. 6 students (including M16) used example feature extractors from the course as-is or with slight modifications, 4 used third-party apps, and 9 wrote their own (often using existing, high-level APIs for specific sensors, such as Leap Motion).

All 5 students modeling real-world phenomena mentioned model accuracy as being important. Additionally, accuracy appeared to be relevant to at least 9 of the 16 students modeling self-defined concepts. For instance, one student (M11) created a virtual "I-Ching Divination" system that must classify which of three types of throwing motions a user has made above a Leap Motion, in order to choose which divination symbols to display. Yet all students' reports revealed that they were attempting to make models that satisfied goals in addition to (or other than) accuracy: these included providing "intuitive" (M10, M16, M21) or "fluid" (M16, M24) control, ensuring that relationships between input and output domains were perceptible to performers or audience (M4, M6, M8, M18, M25), and satisfying other aesthetic criteria (e.g., M16's desire for animation smoothness and M21's desire for playfulness).

ML did not appear to be necessary in all projects. For a handful of projects such as M9's, the final project behavior could arguably be implemented more accurately by explicitly programming model functions. One student first drew a paper sketch of the decision boundaries he wanted in a 2D feature space, then carefully hand-crafted examples to build this classifier, even though these boundaries would have been easy to implement using if-statements. At least two others used regression to simply re-scale one-dimensional data; this can be accomplished with a single function call in many creative coding languages, such as `map()` in Processing.

6.3.4 *Self-reported challenges.* All students had been asked to describe the challenges they encountered in their final report. Very frequently, students described challenges inherent to any creative project, especially—like M9, M16, and M21—deciding what precise form their projects should take in order to reach their high-level design goals (e.g., to make something that felt intuitive or engaging). Additionally, nine mentioned specific challenges related to obtaining or processing inputs; these included choosing features for noisy sensors, and choosing which input device to use. Five mentioned challenges deciding how to process and use model outputs, including deciding whether to apply post-processing (like M16), and simply determining how to write programs that used outputs in an interesting way. Four students (including M16, M21, and the pop song student) explicitly reported improving the accuracy of their ML model(s) to be a challenge.

6.3.5 *Other observed challenges.* Simple feature selection and engineering approaches would have likely improved accuracy (or required fewer training examples) for some students. For instance, using z-axis (height) coordinates of fingers, relative to the palm, would have made M9's tilt model more accurate (compared to using absolute x-, y-, and z-coordinates), and M16 would have benefited from using facial key-point positions relative to the centre of the face (rather than their absolute position). Students had seen these techniques in lecture and could implement them without programming using the InputHelper. It is unclear whether students were not yet comfortable with these techniques, or whether they perceived changing the features as offering too little practical payoff for the amount of extra work required. (Notably, when a Wekinator user opts to change the features used, a new project must be created with new examples, as there is no way to efficiently re-compute features for training examples already recorded.)

Making different choices about the number of models and the precise problem each one was used to solve could also have benefited some students. One such student made a Max/MSP voice harmonizer in which the computer responds to a new sung pitch by a playing a complementary three-note chord. It would have likely been straightforward to implement this using a single classifier, with one class per sung pitch, whose output is used by Max/MSP to select which chord to play. Instead, the student used twelve binary classifiers, each of which was responsible for turning one possible chord pitch on or off. Changing the chord choice for a single pitch therefore required retraining of several classifiers. Also, because each classifier was typically trained to be 'on' for a number of sung pitches, re-training a classifier to produce a different output for a given sung pitch could inadvertently introduce false negative or false positive errors for that classifier when applied to other sung pitches. Structuring projects like this more successfully requires pairing ML knowledge with skill in decomposing a complex implementation project into smaller components—a skill more commonly associated with software engineering than with ML.

6.3.6 Implications for teaching and research. The above analysis provides some insight into into the types of creative work that learning ML can facilitate, and into the ways of using ML that were supported by this curriculum and toolset. Feature engineering and project structuring emerged as challenges that could be addressed with changes to the curriculum. For instance, a lab activity could ask students to sketch out structures for example hypothetical projects on paper, without implementing anything (and therefore being less influenced by what is easy or hard in a given ML tool). This finding has also motivated changes to tools: a new fork of Wekinator is currently being implemented to make it easier for students to visualize and change features without using InputHelper, starting a new project, or re-recording the training examples.

It is unclear why some students used ML in projects that could have been implemented without ML. Did these students not fully grasp the potential of ML to make new types of projects? Was it not relevant to their goals? Or, did using ML still provide some value to them—for instance, by making work with sensors and other data sources more tractable for students who were not advanced programmers (as reported by M21)? Or by making it easier to prototype a wider range of designs than could have been efficiently explored using programming? Tracking students' work beyond the course should help to understand the reasons for this phenomenon. For instance, do students continue to use ML when it is no longer required? Do they stop using ML—or change how they use it—as their programming skills improve?

Although only four students reported challenges in building an accurate model, one cannot conclude that this is because the other students were capable of making ML do everything they wanted. When changing training data through IML fails to improve a model that is not doing what a student wants, they are faced with many options. One option is to reformulate the ML component of the system: reasoning about ML algorithms, features, and data to identify ways to improve the model (i.e., LO7 in Table 2). But students may also reformulate other aspects of the design, such as changing the input device or data source, or changing the type of gestures or actions used for control. For students not committed a priori to modeling a specific phenomenon, changing these other aspects of design might offer a faster and more predictable improvement than spending time debugging an ML model through changes to its feature representation or algorithm. Indeed, most students—like those in Section 6.3.2—changed many aspects of their design over time in response to a number of factors, including finding that they were dissatisfied with the current implementation. These observations carry two implications for teaching: First, the fact that students can creatively "work around" ML failures implies that activities other than the final project are needed to assess students' achievement of the learning objectives in Table 2. Second, given that many students still cared about model accuracy, activities that help support students' skills in reformulating ML are still likely to be useful. As students find it easier to build and reformulate models to satisfy their goals, they may avoid unnecessary and unwanted reformulations of other aspects of their design.

7 DISCUSSION

7.1 Improving Understanding of Creative ML Teaching and Learning

The brief analyses in Section 6 have improved understanding of learner experiences in ways that have driven a number of changes to teaching, as discussed above. Yet there is much more to do. Each analysis considered a small number of students from a single cohort, and especially little is known about the backgrounds of students in the MOOC. Repeating analyses with different cohorts—and examining how student experiences and outcomes differ across cohorts—will be valuable.

Further, Section 4 has presented a number of strategies and tools for teaching creative ML. Evaluation of more of these strategies and tools is now necessary in order to discover whether they are functioning as intended. Crucially, the proposed approaches to explaining algorithms and supporting experiential learning without requiring math and programming rely heavily on 2D visual representations of learning algorithms and models. It has been assumed that the proposed explanations and activities enable students to reason effectively about learning algorithms in two dimensions—e.g., about the types of decision boundaries an algorithm will make, how it will be affected by noise or outliers, how properties of the learning problem may make one algorithm a better choice than another. It has also been assumed that students' ability to reason in two dimensions will be sufficient to enable them to do practical work higher dimensions without too many problems. If these assumptions are true, such approaches to teaching ML could be useful to other populations of students, as well.

It is therefore important to test these assumptions and discover where they break down. Examination of student lab activities from these courses is a possible first step. For instance, assignment A4 (Figure 5) asks students to write a short reflection following the creation of decision boundaries in the Classification Explorer (e.g., noting how sensitive each algorithm is to outliers in the training data, and observing how the model changes as they add more examples). It then asks students to experiment with multiple classification algorithms while building a "classification system you find useful and/or interesting," then to answer some reflective questions about this process (e.g., about the biggest challenge they encountered, and about the advice they would give to another student doing this task). Together, this data should provide some insight into whether working with the Classifier Explorer seemed to help students reason effectively in two dimensions, and into how their reasoning changed when working on an open-ended task in higher dimensions.

Having more scalable ways of evaluating student reasoning would also be valuable to test such assumptions for larger cohorts. For instance, a software version of the "algorithm decision tree" exercise (Section 6.2) could be used to identify common misconceptions, as could simple quizzes. Efficient and validated instruments for assessing students' progress towards mastery of the learning objectives will also enable monitoring how student reasoning evolves in response to different course activities, and how it differs across cohorts.

7.2 Developing an Understanding of PCK for ML

Ko's call for developing an understanding of PCK for ML was influential in motivating this research, and the components of ML PCK that he enumerates have provided a useful structure for framing the contributions of this paper. This paper has proposed a number of representations, analogies, examples, and explanations of ML, as well as methods of informally assessing knowledge of ML concepts. The analyses in Section 6 have informed a better understanding of how these support and reveal student learning, and they have uncovered some of learners' conceptions, mistakes, and challenges.

Yet Ko's list does not include consideration of "why" or "what" to teach—two components of Grossman's [32] formulation of PCK. As Section 2.3 argues, it was important to consider these components as well when motivating, structuring, and beginning to evaluate a curriculum for creative ML, in which the rationale for teaching is so different from more conventional ML teaching. However, I would argue that the questions of "why" and "what" to teach are also important components to consider for ML teaching to other cohorts, including to computer science students. Why should computer science students learn ML? To become ML researchers? To prepare them for jobs that require making accurate models of data? To develop an aesthetic appreciation for a fascinating family of algorithms? Different ML classes taught to different cohorts are likely to entail different answers. The question of what to teach is contingent on these, and also deserving of explicit consideration. For instance, like creative ML students, computer science ML students may benefit from instruction on topics beyond ML itself. This can include instruction on techniques for data processing and visualization, or managing ML at scale using GPUs or distributed computing.

This paper has also considered some aspects of *how* to teach ML that are not well captured by the above components. Most notably, the technology chosen and developed for these courses strongly influenced the ways that concepts were represented and explained in lectures, demos, and lab activities, as well as the scope of lab and project work that could be supported. For instance, Wekinator was chosen in part because it supports IML workflows and experimentation by students who lack strong programming skills. The Classification and Regression Explorer programs were designed to build fluency in reasoning about algorithms in two dimensions, and about how IML approaches can modify models. The example feature extractors were created to enable student experimentation and project work using a wide variety of sensors and input devices. Affordances of these technologies also encouraged certain ways of working at the expense of others. For example, Wekinator encourages creation of real-time projects, it discourages frequent changing of features, and it cannot be used to run ML projects on mobile devices or the web.

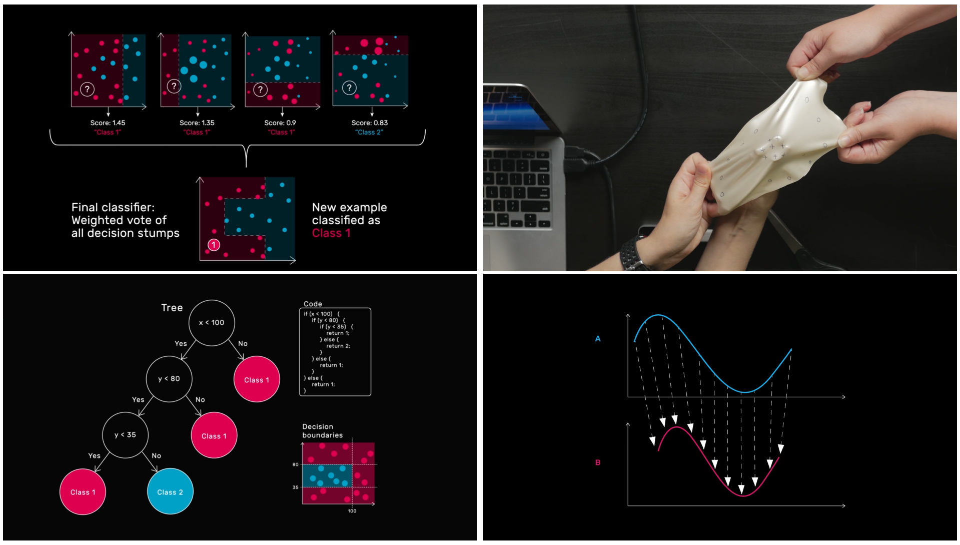
The concept of "technological pedagogical content knowledge" (TPCK) is therefore relevant: Mishra et al. describe TPCK as "the basis of good teaching with technology"; it "requires an understanding of the representations of concepts using technologies; pedagogical techniques that use technologies in constructive ways to teach content; knowledge of what makes concepts difficult or easy to learn and how technology can help redress some of the problems that students face." [50, p. 1029]. These concepts are relevant not only to the structuring and evaluation of the curriculum described here, but to other ML courses as well, where the choice of tools (e.g., ML programming libraries, scaffolding examples created by educators) will also strongly impact how and what students learn. I therefore argue that any conception of PCK for ML should also consider how technology intersects with each of the components in Table 1.

8 CONCLUSIONS

This article has argued for the importance and feasibility of teaching ML to creative practitioners. It has proposed a set of learning objectives and a set of strategies for the effective teaching of creative ML, and it has described how these have been applied within three courses (among the first in the world to address this subject in depth). Inaugural analysis of data from these courses have been used to inform a deeper understanding of why, what, and how to teach creative practitioners about ML. This work provides a number of practical resources—including syllabi, activities, and open-source learning technologies—that can be adopted and adapted by others. It also contributes to a nascent body of research on ML education, helping to deepen understanding of how ML may be taught, and to whom.

A ADDITIONAL EXAMPLES OF VISUAL ILLUSTRATIONS

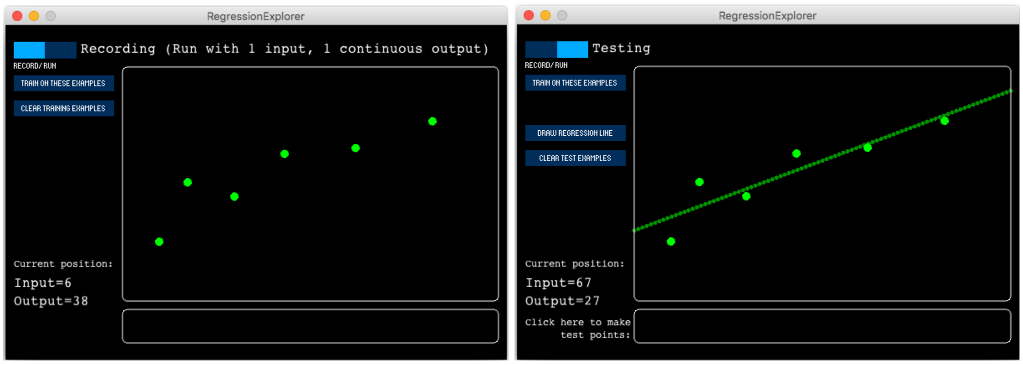
Stills from additional animations used to illustrate algorithms appear below. Clockwise, from top left: AdaBoost, support vector machines, dynamic time warping, decision trees.



```
if (x < 100)  {
    if (y < 80)  {
        if (y < 35)  {
            return 1;
        } else {
            return 2;
        }
    } else {
        return 1;
    }
} else {
    return 1;
}
```

B REGRESSION EXPLORER

The Regression Explorer interface allows users to create new training data by clicking on screen (left) and to evaluate the trained model by clicking to add test points or by automatically generating many test points to illustrate the regression line (right). Here, linear regression is used.



```
Recording (Run with 1 input, 1 continuous output)

Current position:
Input=6
Output=38
```

```
Testing

Current position:
Input=67
Output=27
Click here to make
test points:
```

C MOOC COURSE SYLLABUS

The syllabus for the seven-session MOOC appears below:

MOOC Lectures (L) and Assignments (A) with Learning Objectives (LO) *Assignment marking approach listed as: auto-graded logs (AG), forum participation (F), peer feedback (P)*
L1: Introduction (LO1, LO2, LO5) Course overview. ML Pipeline. Wekinator. Using OSC to pass data to/from Wekinator.
A1: Getting Started with ML (LO1, LO2, LO4, LO5) A1.1: Follow detailed instructions to train and run a very simple ML system using Wekinator with online examples for input (e.g., webcam) and output (e.g., sound or animation). (AG) A1.2: Brainstorm 3 possible creative ML projects and describe on the forum. (F)
L2: Classification, Part 1 (LO1, LO2, LO3, LO4, LO8) What is classification? Nearest-neighbor and Decision Stump algorithms. Artistic applications of classification. What are features? Description and demo of a musical piece using classification.
A2: Creating Classifiers (LO1, LO2, LO3, LO4, LO5, LO6, LO7) A2.1: Engage in free-form experimentation with Classification Explorer, observing how decision boundaries change with different training sets and algorithms. (AG) A2.2: Generate training sets to re-create specific decision boundaries in Classification Explorer using kNN and Decision Stumps. (AG) A2.3: Use Wekinator and a real-time input of your choice to create an accurately controllable classifier, then demonstrate yourself using it to produce a specific sequence of class outputs. (AG) A2.4: Post a question, insight, or other contribution to the forum. (F)
L3: Regression (LO1, LO2, LO3, LO4, LO8) What is regression? Linear and polynomial regression. What makes a "good" regression model? Intro to neural networks. Using regression to create "mappings" for new expressive controllers. Description and demo of a musical instrument that uses regression for mapping. Practical tips for using regression with Wekinator. *Optional additional content:* Training as solving an optimisation problem; overview of neural networks training procedure.
A3: Regression (LO1, LO2, LO3, LO4, LO5, LO6, LO7) A3.1: Engage in free-form experimentation with Regression Explorer, observing how regression lines change with different training sets and algorithms. (AG) A3.2: Generate data points to re-create specific regression lines using Regression Explorer with linear and polynomial regression. (AG) A3.3: Use Wekinator and a real-time input of your choice to create an accurately controllable regression model, then demonstrate yourself using it to produce a specific sequence of regression outputs. (AG) A3.4: Post a question, insight, or other contribution to the forum. (F)
L4: Classification, Part 2 (LO1, LO2, LO3, LO4, LO6, LO7) What is a "good" classifier? Reasoning about feature spaces. Naive Bayes. Decision stumps and decision trees. AdaBoost. SVMs. Evaluating classifiers using training accuracy, cross-validation. Using probability distributions over classes (why it can be useful, how to do it in Wekinator). Using more than one classifier at once in a project (why this can be useful, how to do it in Wekinator).
A4: More Classification Fun (LO1, LO2, LO3, LO4, LO5, LO6, LO7) (See Figure 5) A4.1: Generate data points to re-create specific classification boundaries using Classification Explorer with kNN, Decision Trees, AdaBoost, and SVM. (AG) A4.2: Use Wekinator and any real-time input of your choice to "build a classification system that you find useful and/or interesting." Spend at least 30 minutes experimenting, then answer some reflective questions (e.g., "What was the biggest challenge you encountered?" "How successful were you in building a classifier that worked the way you wanted?" "If you were to give advice to another student trying to build a similar system, what advice would you give?" (AG) A4.3: Post a question, insight, the answers to your reflective questions, or other contribution to the forum. (F)

L5: Sensors and Features: Generating Useful Inputs for ML (LO1, LO2, LO3, LO7) Simple features. Basic feature engineering principles and strategies (e.g., normalising, smoothing, segmentation). Common audio and video analysis features. Demos of feature engineering using Input Helper for projects using audio and sensor inputs.
A5: Creating Your Own Feature Extractor (LO1, LO2, LO3, LO4, LO5, LO6, LO7) A5.1: Choose one of the following: *Option A:* Implement a useful feature extractor in the environment of your choosing, then post it to the forum with instructions on how to run it. (AG, F) *Option B:* Use Input Helper to create a useful modification of features from one of the input examples, then post your Input Helper project to the forum with instructions on how to run it (AG, F) A5.2: Choose 3 other students' feature extractors from A.5.1 and write each of them 150 words of constructive feedback. (P)
L6: Working with Time (LO1, LO2, LO3, LO4, LO6, LO7, LO8) Using features that encode change over time. Dynamic time warping [53]. Hidden Markov Models. Gesture Follower [5] and Gesture Variation Follower (GVF) [13]. Demo using GVF to design sonic interactions. Designing custom algorithms for music.
A6: Dynamic Time Warping (LO1, LO2, LO3, LO4, LO5, LO6, LO7) A6.1: Train DTW Explorer to distinguish between 4 shapes drawn with mouse (AG) A6.2: Build a DTW system that "you find useful and/or interesting." Answer questions similar to A4.2. (AG)
L7: Developing a Practice with ML; Wrap-up (LO1, LO2, LO8) Overview of other topics (unsupervised learning, computational creativity, deep learning). How is ML in the arts different from more conventional domains? Guest lecture by well-known composer discussing how and why she uses ML in projects (e.g., how she structures projects, how she goes about training models, what advantages ML brings to her work).
A7: Final Creative Project (LO1–LO8) A7.1: Make a creative project with Wekinator. Answer some reflective questions about it. Write instructions so that someone else (who has the required hardware/software) could run your project. Optionally create a demo video. (AG) A7.2: Provide constructive written feedback on at least 3 other students' final projects. (P)

D UNDERGRADUATE COURSE SYLLABUS

The undergraduate syllabus makes use of lectures (L) and assignments (A) from the MOOC (Appendix C). The undergraduate syllabus also includes some content about deep learning and generative methods (Weeks 5, 7, 8) that is not discussed in this article.

Before-Class (BC) and In-Class (IC) Learning Activities in Undergraduate Course
with learning objectives (LO) listed for activities not included in the MOOC

Week	
1.	BC: Watch L1 IC: A1.1 (in pairs), brainstorm creative applications of supervised learning in small groups (LO1, LO2)
2.	BC: Watch L2 IC: A2.1, A2.2, A2.3 (in pairs)
3.	BC: Watch L3 IC: A3.1, A3.2, A3.3 (in pairs). Draw "algorithm decision tree" (Section 6.2) individually, then discuss in small groups (LO3). Additional creative project brainstorming in small groups (LO2).
4.	BC: Watch L4 IC: A4.1, A4.2 (in pairs). In small groups, come up with 1-sentence descriptions of each learning algorithm studied so far (LO3).
5.	BC: Update "algorithm decision tree" (LO3). Watch very short video about creative deep learning. IC: Small group discussion and refinement of algorithm decision tree (LO3). Work in groups to experiment with sensors: acquiring and plotting data from Arduino, Leap Motion, WiiMotes (LO2, LO3, LO5). Optionally discuss advanced topics with staff (biosignals, deep learning).
6.	BC: Watch L5 IC: Work in small groups to use Input Helper and a data visualizer to explore feature engineering for sensors and game controllers, then answer reflective questions about features; build an end-to-end system with sensors or controller (LO2, LO3, LO4, LO5, LO7).
7.	BC: Watch L6, and optionally an online lecture about creative deep learning IC: A6.1, A6.2 (in pairs)
8.	BC: Option 1: Install TensorFlow. Option 2: Read through web tutorials on Markov generation. IC: Option 1: Experimentation in pairs with TensorFlow for image generation. Option 2: Experimentation in pairs with web-based content generation tools (no programming).
9.	BC: L7; Find an existing creative project that uses ML that you find inspiring, and post it to the forum (LO8). IC: Small group discussion of inspiring projects (LO8). Peer feedback on final project ideas (LO2).
10.	BC: Work on final project. IC: Work on final project in class.

ACKNOWLEDGMENTS

I would like to thank Kadenze Inc. for the opportunity to teach Machine Learning for Musicians and Artists on their platform, and for the immeasurable support Kadenze personnel have offered in the creation and running of the course, and in enabling me to access course data. I am also thankful for the assistance that Louis McCallum, Kiona Niehaus, and Esben Sørig provided in coding the data. Additionally, I am deeply grateful to to the TOCE reviewers and guest editor for their expert feedback and insightful suggestions for improving this article.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. TensorFlow: A System for Large-Scale Machine Learning. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)*, Vol. 16. 265–283.

[2] Charles Ames. 1989. The Markov Process as a Compositional Model: A Survey and Tutorial. *Leonardo* 22, 2 (1989), 175–187.

[3] Mordechai Ben-Ari. 1998. Constructivism in computer science education. In *ACM SIGCSE Bulletin'*, Vol. 30. ACM, 257–261.

[4] Francisco Bernardo, Mick Grierson, and Rebecca Fiebrink. 2018. User-Centred Design Actions for Lightweight Evaluation of an Interactive Machine Learning Toolkit. *Journal of Science and Technology of the Arts* 10, 2 (2018), 2–25.

[5] Frédéric Bevilacqua, Bruno Zamborlin, Anthony Sypniewski, Norbert Schnell, Fabrice Guédy, and Nicolas Rasamimanana. 2009. Continuous realtime gesture following and recognition. In *International gesture workshop*. Springer, 73–84.

[6] Jacob Lowell Bishop and Matthew A Verleger. 2013. The flipped classroom: A survey of the research. In *ASEE National Conference Proceedings, Atlanta, GA*, Vol. 30. 1–18.

[7] Blender. 2018. blender.org - Home of the Blender project - Free and Open 3D Creation Software. (2018). https://www.blender.org/ Accessed: 2018-10-15.

[8] Thomas Both. 2013. *Design Thinking Bootleg*. Hasso Plattner Institute of Design at Stanford, Stanford, CA. https://dschool.stanford.edu/resources/the-bootcamp-bootleg

[9] Mason Bretan, Sageev Oore, Jesse Engel, Douglas Eck, and Larry Heck. 2017. Deep Music: Towards Musical Dialogue.. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)*. 5081–5082.

[10] Jamie Bullock and Ali Momeni. 2015. Ml.lib: Robust, cross-platform, open-source machine learning for Max and PureData. In *Proceedings of the International Conference on New Interfaces for Musical Expression*. 265–270.

[11] C.J.C. Burges. 1998. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery* 2, 2 (1998), 121–167.

[12] Antonio Camurri, Shuji Hashimoto, Matteo Ricchetti, Andrea Ricci, Kenji Suzuki, Riccardo Trocca, and Gualtiero Volpe. 2000. Eyesweb: Toward gesture and affect recognition in interactive dance and music systems. *Computer Music Journal* 24, 1 (2000), 57–69.

[13] Baptiste Caramiaux, Nicola Montecchio, Atau Tanaka, and Frédéric Bevilacqua. 2015. Adaptive gesture recognition with variation estimation for interactive systems. *ACM Transactions on Interactive Intelligent Systems (TiiS)* 4, 4 (2015), 18.

[14] Alex J. Champandard. 2016. Semantic Style Transfer and Turning Two-Bit Doodles into Fine Artworks. *CoRR* abs/1603.01768 (2016). arXiv:1603.01768 http://arxiv.org/abs/1603.01768

[15] Louis Cohen, Lawrence Manion, and Keith Morrison. 2011. *Research Methods in Education*. Routledge, Abingdon, UK.

[16] Luke Dahl, Jorge Herrera, and Carr Wilkerson. 2011. TweetDreams: Making Music with the Audience and the World using Real-time Twitter Data.. In *Proceedings of the International Conference on New Interfaces for Musical Expression*. 272–275.

[17] John Dewey. 1938. *Experience & Education*. Kappa Delta Pi, New York.

[18] Charles Dodge. 1970. Earth's Magnetic Field: Realizations in Computed Electronic Sound. (1970). Audio Recording, Nonesuch/Elektra 71250.

[19] Daniel C Edelson, Douglas N Gordin, and Roy D Pea. 1999. Addressing the challenges of inquiry-based learning through technology and curriculum design. *Journal of the learning sciences* 8, 3-4 (1999), 391–450.

[20] Jerry Alan Fails and Dan R Olsen Jr. 2003. Interactive machine learning. In *Proceedings of the 8th international conference on Intelligent user interfaces*. ACM, 39–45.

[21] Rebecca Fiebrink. 2011. *Real-time human interaction with supervised learning algorithms for music composition and performance*. PhD Thesis, Princeton University.

[22] Rebecca Fiebrink. 2018. Machine Learning for Musicians and Artists (online course)). (2018). https://www.kadenze.com/courses/machine-learning-for-musicians-and-artists/info Accessed: 2018-10-01.

[23] Rebecca Fiebrink. 2018. The Wekinator: Software for real-time, interactive machine learning. (2018). http://www.wekinator.org/

[24] Rebecca Fiebrink, Perry R Cook, and Dan Trueman. 2011. Human model evaluation in interactive supervised learning. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 147–156.

[25] Rebecca Fiebrink, Dan Trueman, and Perry R Cook. 2009. A Meta-Instrument for Interactive, On-the-Fly Machine Learning. In *Proceedings of the International Conference on New Interfaces for Musical Expression*. 280–285.

[26] The Processing Foundation. 2018. The Processing Foundation. (2018). https://processingfoundation.org/ Accessed: 2018-04-06.

[27] Jason Freeman, Brian Magerko, Tom McKlin, Mike Reilly, Justin Permar, Cameron Summers, and Eric Fruchter. 2014. Engaging underrepresented groups in high school introductory computing through computational remixing with EarSketch. In *Proceedings of the 45th ACM technical symposium on Computer science education*. ACM, 85–90.

[28] Yoav Freund and Robert E. Schapire. 1997. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *J. Comput. System Sci.* 55, 1 (1997), 119–139.

[29] Nicholas Edward Gillian and Joseph A Paradiso. 2014. The gesture recognition toolkit. *Journal of Machine Learning Research* 15, 1 (2014), 3483–3487.

[30] Nicholas Edward Gillian and Joseph A Paradiso. 2014. The gesture recognition toolkit. *Journal of Machine Learning Research* 15, 1 (2014), 3483–3487.

[31] Google. 2018. Machine Learning Crash Course with TensorFlow APIs. (2018). https://developers.google.com/machine-learning/crash-course/ Accessed: 2018-06-15.

[32] Pamela Lynn Grossman. 1990. *The making of a teacher: Teacher knowledge and teacher education.* Teachers College Press, Teachers College, Columbia University.

[33] Idit Ed Harel and Seymour Ed Papert. 1991. *Constructionism.* Ablex Publishing.

[34] Thomas J Howard, Stephen J Culley, and Elies Dekoninck. 2008. Describing the creative design process by the integration of engineering design and cognitive psychology literature. *Design studies* 29, 2 (2008), 160–180.

[35] Aleata Hubbard. 2018. Pedagogical content knowledge in computing education: A review of the research literature. *Computer Science Education* 28, 2 (2018), 117–135. https://doi.org/10.1080/08993408.2018.1509580

[36] Andrew Hugill and Hongji Yang. 2013. The creative turn: New challenges for computing. *International Journal of Creative Computing* 1, 1 (2013), 4–19.

[37] Kadenze Inc. 2018. Online STEAM Courses From Top Universities : Kadenze. (2018). www.kadenze.com Accessed: 2018-06-15.

[38] NYU ITP. 2018. ML5js: Friendly Machine Learning for the Web. (2018). https://ml5js.org/ Accessed: 2018-06-15.

[39] Giulio Jacucci, Anna Spagnolli, Alessandro Chalambalakis, Ann Morrison, Lassi Liikkanen, Stefano Roveda, and Massimo Bertoncini. 2009. Bodily explorations in space: Social experience of a multimodal art installation. In *Proceedings of the International Federation for Information Processing Conference on Human-Computer Interaction.* Springer, 62–75.

[40] Katy Jordan. 2015. Massive open online course completion rates revisited: Assessment, length and attrition. *The International Review of Research in Open and Distributed Learning* 16, 3 (2015).

[41] René F Kizilcec, Chris Piech, and Emily Schneider. 2013. Deconstructing disengagement: analyzing learner subpopulations in massive open online courses. In *Proceedings of the third international conference on learning analytics and knowledge.* ACM, 170–179.

[42] A. J. Ko. 2017. We need to learn how to teach machine learning. *Blog post. 21 August 2017.* (2017). https://medium.com/bits-and-behavior/we-need-to-learn-how-to-teach-machine-learning-acc78bac3ff8

[43] David A Kolb. 1984. *Experiential learning: Experience as the source of learning and development.* Prentice Hall, Englewood Cliffs, NJ, USA.

[44] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems.* 1097–1105.

[45] Ryan Laney. 2017. Syntien: OSC Control Surface. iOS Store https://itunes.apple.com/us/app/syntien/id1203153534?mt=8. (2017).

[46] John Lansdown. 1989. Generative techniques in graphical computer art: Some possibilities and practices. In *Computers in art, design and animation.* Springer, 56–79.

[47] M Lee, A. Freed, and D. Wessel. 1991. Real-Time Neural Network Processing of Gestural and Acoustic Signals. In *Proceedings of the International Computer Music Conference (ICMC).* 277–280.

[48] Beth Logan. 2000. Mel Frequency Cepstral Coefficients for Music Modeling. In *Proceedings of the International Symposium on Music Information Retrieval.*

[49] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, Doris Xin, Reynold Xin, Michael J. Franklin, Reza Zadeh, Matei Zaharia, and Ameet Talwalkar. 2016. MLlib: Machine Learning in Apache Spark. *Journal of Machine Learning Research* 17, 34 (2016), 1–7.

[50] Punya Mishra and Matthew J Koehler. 2006. Technological pedagogical content knowledge: A framework for teacher knowledge. *Teachers college record* 108, 6 (2006), 1017.

[51] Luis C. Moll, Cathy Amanti, Deborah Neff, and Norma Gonzalez. 1992. Funds of Knowledge for Teaching: Using a Qualitative Approach to Connect Homes and Classrooms. *Theory Into Practice* 31, 2 (1992).

[52] Alexander Mordvintsev, Christopher Olah, and Mike Tyka. 2015. Inceptionism: Going deeper into neural networks. *Google Research Blog. Retrieved 6 April 2018.* (2015). https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html

[53] Meinard Müller. 2007. *Dynamic Time Warping.* Springer-Verlag Berlin Heidelberg, 69–84.

[54] François Pachet. 2006. Enhancing individual creativity with interactive musical reflexive systems. *Musical creativity* (2006), 359.

[55] Kayur Patel, Naomi Bancroft, Steven M Drucker, James Fogarty, Andrew J Ko, and James Landay. 2010. Gestalt: Integrated support for implementation and analysis in machine learning. In *Proceedings of the 23rd Annual ACM Symposium on User Interface Software and Technology (UIST).* ACM, 37–46.

[56] Kayur Dushyant Patel. 2012. *Lowering the barrier to applying machine learning.* PhD Thesis, University of Washington.

[57] Roy D Pea. 1986. Language-independent conceptual “bugs” in novice programming. *Journal of Educational Computing Research* 2, 1 (1986), 25–36.

[58] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *Journal of machine learning research* 12, Oct (2011), 2825–2830.

[59] K. Peppler. 2013. STEAM-Powered Computing Education: Using E-Textiles to Integrate the Arts and STEM. *Computer* 46, 9 (September 2013), 38–43.

[60] Libby Plummer. 2017. The next big act out of Abbey Road could be an AI startup. *Wired* (19 September 2017). https://www.wired.co.uk/article/abbey-road-red-startups-ai-music

[61] Christopher Raphael. 2010. Music Plus One and Machine Learning. In *Proceedings of the Twenty-Seventh International Conference on Machine Learning (ICML)*.

[62] Mitchel Resnick, Brad Myers, Kumiyo Nakakoji, Ben Shneiderman, Randy Pausch, Ted Selker, and Mike Eisenberg. 2005. Design Principles for Tools to Support Creative Thinking. In *Report of Workshop on Creativity Support Tools*. Washington, DC, USA.

[63] Stuart Russell and Peter Norvig. 2016. *Artificial Intelligence: A Modern Approach* (3rd ed.). Pearson, Chapter 18.

[64] Rob Schapire. 2014. COS 511: Theoretical Machine Learning. (2014). https://www.cs.princeton.edu/courses/archive/spring14/cos511/scribe_notes/0204.pdf Accessed: 2018-06-15.

[65] Janelle Shane. 2018. We asked a neural network to bake us a cake. The results were...interesting. *Popular Science* (29 March 2018). https://www.popsci.com/neural-network-bakes-a-cake

[66] Lee S Shulman. 1986. Those who understand: Knowledge growth in teaching. *Educational researcher* 15, 2 (1986), 4–14.

[67] Tom Simonite. 2017. A ‘Neurographer’ Puts the Art in Artificial Intelligence. *Wired* (6 July 2017). https://www.wired.com/story/neurographer-puts-the-art-in-artificial-intelligence/

[68] Laetitia Sonami. 1991. Lady’s Glove. (1991). http://sonami.net/ladys-glove/ Accessed: 2018-06-15.

[69] Juha Sorva. 2013. Notional machines and introductory programming education. *ACM Transactions on Computing Education (TOCE)* 13, 2 (2013), 8.

[70] TechSpark. 2017. Guest blog: How one UWE Bristol researcher’s work led to AI gloves that control sound. (2017). https://techspark.co/guest-blog-uwe-bristols-ai-gloves-control-sound-without-need-laptop-mouse/ Accessed: 2018-06-15.

[71] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W. Senior, and Koray Kavukcuoglu. 2016. WaveNet: A Generative Model for Raw Audio. *CoRR* abs/1609.03499 (2016). arXiv:1609.03499 http://arxiv.org/abs/1609.03499

[72] M. Wright and Adrian Freed. 1997. Open Sound Control: A New Protocol for Communicating with Sound Synthesizers. In *Proceedings of the International Computer Music Conference (ICMC)*.

[73] Matthew John Yee-King, Mick Grierson, and Mark d’Inverno. 2017. Evidencing the Value of Inquiry Based, Constructionist Learning for Student Coders. *International Journal of Engineering Pedagogy (IJEP)* 7, 3 (2017), 109–129.